LA PRIMERA REVISTA DE PROGRAMACIÓN EN CASTELLANO

SOLO PROGRAMADORES

O VI. 2ª ÉPOCA NÚMERO 66 UNA PUBLICACIÓN DE: REVISTAS PROFESIONALES S.L. 975 Ptas. • 995 Escudos portugueses • 5,86 € (IVA incluido)

WML: CREAR WEBS PARA MÓVILES

VISUAL BASIC
Programación Orientada a Objetos: Novedades de VB 7

HERRAMIENTAS
VisualCafé 4 vs. JBuilder 3

LINUX
MP3 en nuestros programas

ENTORNOS DE DESARROLLO
Visual C++ y MCF

BASES DE DATOS
Aplicaciones de bases de datos con Delphi

PROGRAMACIÓN WEB
Aplicaciones Java con acceso a BBDD

JAVA
Firmado de Applets

DELPHI
Programación OpenGL

8 413042 303299

CONTENIDO CD

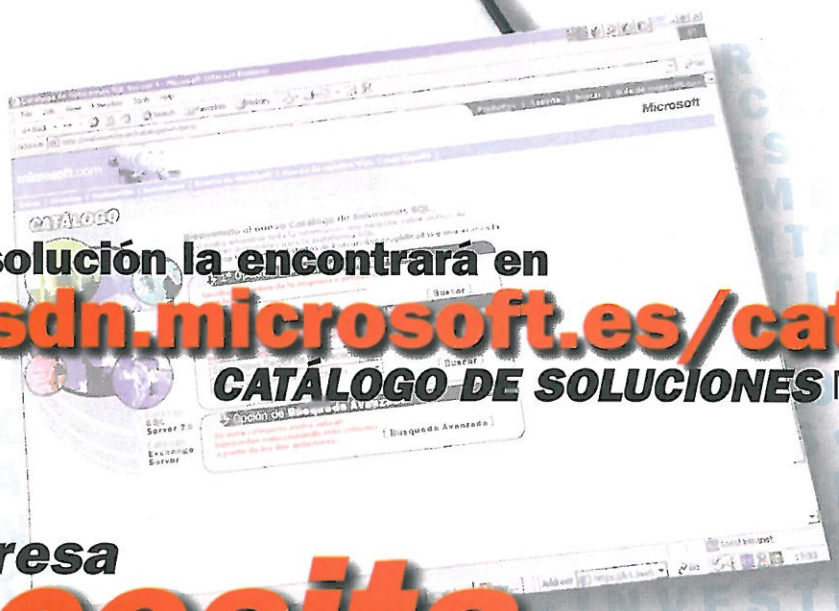
- Visual Perl Editor 2.5
- BinaryWork MP3 OCX 2.0.0
- 3DCharting Toolkit 4.0
- DocuClass 1.0
- JQuickCompile 1.0
- ZoneAlarm 2.0.26
- ProDelphi 7.5
- Morfit 3D Engine SDK
- WinEoute Pro 4.1a
- WebSphere Studio 3.0

Su empresa

Tiene

Aplicaciones desarrolladas bajo Microsoft Windows

"Nuestra empresa ha desarrollado una aplicación. ¿Cuál será el sitio donde la puedan conocer el mayor número de clientes potenciales?"



La solución la encontrará en

msdn.microsoft.es/catalogo

CATÁLOGO DE SOLUCIONES Microsoft **Windows**

El lugar donde se encuentra el mayor número de aplicaciones desarrolladas bajo Microsoft Windows para cualquier tipo de empresas.

Su empresa

necesita

Aplicaciones de desarrollo bajo Microsoft Windows

"Buscamos una aplicación de gestión a medida para la actividad de nuestra empresa"

Las 50 primeras soluciones* que se incluyan en nuestro catálogo de soluciones, obtendrán un ratón IntelliMouse Explorer o un teclado Microsoft Internet Keyboard

* Solamente se obtendrá un regalo por empresa

Microsoft

SÓLO PROGRAMADORES

Número 66
SÓLO PROGRAMADORES
es una publicación de
REVISTAS PROFESIONALES S.L.

Editor
Agustín G. Buelta
Director
Javier Amado Buiza
Coordinador Técnico
Eduardo De Riquer Frutos
Coordinadoras de Redacción
Gema Romero Moreno-Manzanaro
Cristina Peña del Pozo

Colaboradores
Constantino Sánchez, Juan Luis Ceada, Javier Sanz, Adolfo Aladro, Enrique de la Lastra, Jordi Agost, Vicente A. Sánchez Werner
Maquetación y Tratamiento de Imagen
Paco Risco
Consultas técnicas
atecnica@virtualsw.es

Publicidad
Tel.: (91) 304 78 46
Mariano Sánchez (Barcelona)
Tel.: (93) 322 12 38
Pepín Gallardo (Barcelona)
Tel.: 617 09 36 68

Suscripciones
Rosa Tabares
Tel. (91) 304 87 64 Fax: (91) 327 13 03

Preimpresión
Grebe
Impresión
I. de Impresión
Distribución
Motorpress Ibérica

GRUPO MPI Motorpress-Ibérica

La revista Sólo Programadores no tiene por qué estar de acuerdo con las opiniones escritas por sus colaboradores en los artículos firmados. El editor prohibe expresamente la reproducción total o parcial de los contenidos de la revista sin su autorización escrita.

Depósito legal: M-26827-1994
ISSN: 1134-4792
PRINTED IN SPAIN
COPYRIGHT 30-6-2000
Precio en Canarias, Ceuta y Melilla:
938 ptas. sin I.V.A.
Con sobretasa aérea: 975 ptas.

AEPP
Asociación Española de Editoriales de Publicaciones Periódicas

EDITORIAL

Bienvenidos sean los nuevos teléfonos móviles

Ahora las tecnologías de telefonía móvil y de *Internet* están convergiendo. Podemos decir que el acceso a móvil se está combinando con los servicios basados en el Protocolo de *Internet* (*IP*) lo cual implica nuevas formas de comunicarse, de acceder a la información, de hacer negocios, de aprender e incluso de disfrutar de nuestro tiempo libre.

Por otra parte los analistas del sector predicen que en pocos años existirán mil millones de teléfonos móviles en el mundo, de los cuales más de veinte millones estarán en manos de usuarios españoles, y que muchos de estos teléfonos móviles contarán con algunas de las posibilidades de los ordenadores personales.

Ante estas perspectivas los teléfonos móviles para *Internet* o mejor dicho el Protocolo para Redes Inalámbricas, *WAP*, se está introduciendo en la escena empresarial y en poco tiempo se generará una importante demanda, ya que la posibilidad de acceder al correo electrónico o información de una página *Web* desde un móvil va a transformar la forma de trabajar de muchas personas.

Por esta razón hemos creído importante iniciar una serie de artículos sobre el lenguaje *WML* que permite crear páginas *Web* para los móviles de la próxima generación de los que existen ya algunos modelos de fabricantes muy conocidos. Ni que decir tiene que en este momento existe una gran demanda por parte de las compañías telefónicas de empresas y profesionales que puedan implementar estas nuevas tecnologías. Esperamos pues que esta serie de artículos sea de gran interés y esperamos vuestros comentarios sobre este tema de cara a los siguientes artículos.

Por otra parte seguimos atentos a los últimos desarrollos en lenguajes de programación y ofrecemos en este número, por una parte un amplio análisis sobre las importantes novedades que incorpora la versión 7 de la popular herramienta *Microsoft Visual Básic*, y por otro de lado un examen en profundidad de la cuarta versión de *Visual Café* de *Symantec*, que presenta también un importante abanico de nuevas posibilidades para los programadores que desean generar código *Java Puro* 100 por 100. Y por supuesto incluimos las secciones y series de artículos habituales.

En este momento estamos ya trabajando intensamente para el siguiente número de esta revista que tiene por objeto, como sabéis, que nuestros lectores dispongan de las últimas herramientas de programación y conozcan la mejor manera de emplearlas. Hasta el próximo mes y que disfrutéis leyendo este número de *Sólo Programadores*.

SUMARIO 66

SÓLO PROGRAMADORES

6 Noticias

El mundo de la programación está sufriendo cambios revolucionarios. Por ello no debéis dejar pasar la oportunidad de leer con atención las novedades de las que os informamos desde estas páginas.

9 Contenido del CD-ROM

Como ya es habitual en nuestra revista os regalamos un *CD* en el que podréis encontrar los listados y fuentes de todos los artículos, junto con los mejores programas y las actualizaciones más importantes. Este mes destacamos: *Visual Perl Editor* 2.5, *ZoneAlarm* 2.0.26, *Morfit 3D Engine SDK* y *WinEoute Pro* 4.1a.

22 Java

FIRMADO DE APPLETS (y II)

En la entrega anterior se hizo una primera aproximación al firmado de *Applets*. En esta ocasión acudimos a los aspectos prácticos y aprenderéis cómo utilizar los *applets* con los dos navegadores más importantes del momento: *Internet Explorer* y *Netscape Communicator*. Como comprobaréis, la firma digital varía mucho dependiendo del navegador que empleemos.

12 Portada Telecomunicaciones

CURSO DE PROGRAMACIÓN CON WAP (I)

La tecnología avanza cada día más. Por eso os presentamos una tecnología que empieza a hacer furor. Se trata de *WAP*. Para este nuevo soporte utilizaremos el lenguaje *WML* para crear páginas *Web* destinadas a los móviles de próxima generación. Una nueva serie con la que aprovecharéis la potencia y similitud del navegador *WAP* con los navegadores de páginas *Web*. Con esta tecnología se dejan de poner límites a los usuarios de móviles, que cada día son más numerosos.

60 Visual Basic

PROGRAMACIÓN ORIENTADA A OBJETOS (I): LAS NOVEDADES DE VB 7

Visual Basic 7 ha incluido numerosas mejoras en el área de la orientación a objetos y también modernas características adicionales propias de los lenguajes más modernos. Por lo que empezamos esta serie comentando sus novedades. En posteriores entregas examinaremos las posibilidades que actualmente tenemos en el mercado sobre la programación orientada a objetos.

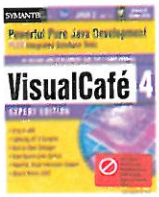
30 Delphi

PROGRAMACIÓN OPENGL EN DELPHI (I)

Esta será la primera entrega dentro del completo mundo de la programación gráfica, centrada en *OpenGL*. Os enseñaremos a trabajar con imágenes *3D*, explotando al máximo las capacidades de aceleración gráfica de nuestras tarjetas de vídeo. La interfaz *OpenGL* está avalada por *Silicon Graphics* y como podréis comprobar dan la talla como expertos en la manipulación de gráficos en *3D*.

38 Herramientas

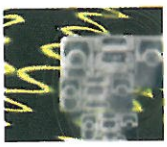
VISUALCAFÉ 4



Presentamos la cuarta versión de *VisualCafé*. Un producto de la casa *Symantec* que viene cargado de novedades para aprovechar de forma sencilla todas las ventajas profesionales que nos ofrece *Java*. Una nueva versión que, con sus tres ediciones, está enfocada directamente al desarrollo de las aplicaciones, generando código *Java Puro* 100%, asegurando así la compatibilidad entre diferentes plataformas.

46 Linux

MP3 EN LINUX (II): AÑADIENDO REPRODUCCIÓN MP3 A NUESTROS PROGRAMAS



Una de las prestaciones más interesantes del formato *MP3* es la posibilidad de incorporar música a nuestros programas utilizando muy poco espacio de disco. Algo que veremos en esta ocasión. Aunque antes tendremos que tener claro qué deseamos hacer con los *MP3* y qué clase de control requiere su uso.

80 Ayuda al lector

No olvidéis consultar esta sección dedicada a vosotros, nuestros lectores. Si nuestras respuestas os resultan ambiguas o poco comprensibles, o simplemente queréis saber más sobre el problema a solucionar, recordad que siempre podréis acudir a nuestra dirección de *E-mail:* solop@virtualsw.es.

54 Bases de Datos

BASES DE DATOS EN DELPHI 5 (III)

En la entrega anterior nos ocupamos de los componentes de acceso y la edición de datos. En este número ponemos en práctica los conocimientos adquiridos con una aplicación práctica y destacamos aquellos aspectos teóricos que habían quedado pendientes, para que el seguimiento de los ejemplos resulte más sencillo.

66 Entornos de Desarrollo

VISUAL C++ Y MFC (III)

Tras presentar los conceptos básicos de la programación con *Visual C++* y *MFC*, llega el momento de adentrarnos en el control de la información dentro de las cajas de texto incluidas en las ventanas o formularios. Utilizaremos controles, cuadros de lista, funciones, códigos y tablas que nos ayudarán en nuestro proyecto.

72 Programación Web

APLICACIONES WEB CON ACCESO A BBDD BASADAS EN JAVA (IV)

Es el momento de entrar en el mundo de la *Web*, sus entornos y aplicaciones dentro del mundo de la programación. A lo largo de este artículo veremos cómo se pueden utilizar diferentes configuraciones para que nuestra aplicación *Web* funcione de manera cómoda y eficiente.

78 Libros

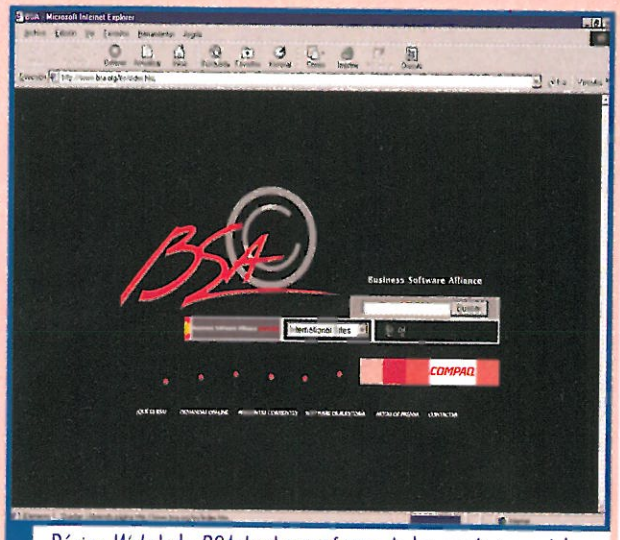
Las últimas novedades sobre programación y los libros que nos han parecido más interesantes aparecen aquí en esta sección. Como ya es habitual, os informamos de la editorial que lo publica, los autores, el idioma y el precio con el que el libro sale al mercado. Esperamos que esta referencia os sea de gran utilidad a la hora de adquirirlos.

LAS DENUNCIAS A LA BSA AUMENTAN UN 350%

El pasado 1 de febrero la *Business Software Alliance* (*BSA*), en su lucha contra la piratería informática, ofreció recompensas económicas de hasta 500.000 pesetas por denuncias de usos fraudulentos de *software*. Desde entonces los contactos con esta asociación han aumentado un 350% con respecto al mes de enero.

De todas las llamadas telefónicas recibidas, de las visitas a la página *Web* www.bsa.org/es y del correo ordinario, la *BSA* ha recibido un total de 107 denuncias en firme por piratería informática. Sin embargo es ahora cuando empieza el verdadero trabajo de la *BSA*, comprobar los datos y si existen indicios de delitos, según el nuevo Código Penal, informar a las Fuerzas de Seguridad del Estado o poner una denuncia ante los tribunales.

El denunciante recibirá el 10% de la indemnización que se imponga a la empresa infractora hasta un máximo de 500.000 pesetas. Un programa de recompensas económicas que estará vigente hasta el 31 de marzo.



Página *Web* de la *BSA* donde se informa de las condiciones del Programa de Recompensas Económicas.

EXPOSICIÓN Y CONFERENCIAS SOBRE LINUX EN MADRID

Para responder al desarrollo que está experimentando el mercado *Linux* en nuestro país *Sky Events* ha organizado la primera feria específica sobre *Linux* que incluirá a los desarrolladores y a los usuarios. Una feria que se celebrará los días 26 y 27 del próximo mes de abril en el Palacio de Congresos de Madrid.

A esta reunión conocida como *Linux Expo* acudirán más de 70 expositores (productores, distribuidores, editores, integradores, y todos los actores del mundo *Linux*). Allí se encontrará representada toda la oferta de soluciones *Linux* y de programación libre unida a un ciclo de conferencias y tutoriales de los mejores especialistas europeos.

La administración del sistema, la seguridad en *Linux*, las aplicaciones para *Internet* o el desarrollo de interfaces gráficas son algunos de los temas que tratarán estas jornadas abiertas a todos los que quieran acercarse a esta primera exposición.

3.ª GENERACIÓN DE LA EMPRESA EN INTERNET

Paul Otellini, vicepresidente ejecutivo y director general del *Architecture Business Group* de *Intel* anunció, en el *Intel Developer Forum* – el principal foro técnico de *Intel*- dirigido a 2000 desarrolladores de aplicaciones *hardware* y *software* basadas en *Intel* que la tercera generación de la empresa en *Internet* es una realidad.

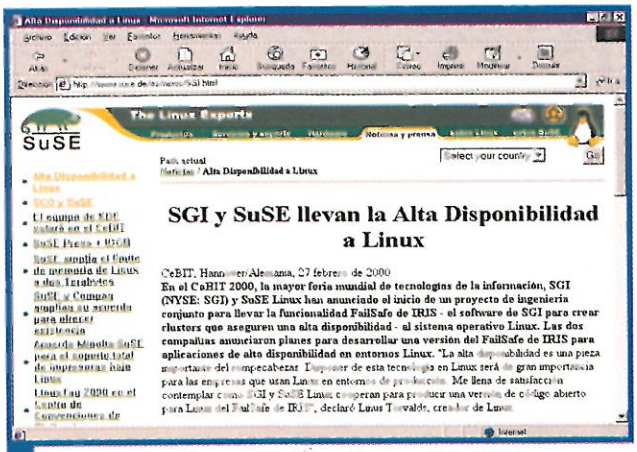
Una nueva generación en la que las empresas gestionarán, a través de la red de redes, sus relaciones con los proveedores de forma electrónica, lo que supone un paso eficaz en las relaciones en tiempo real. Este hecho supone una novedad con respecto a épocas no muy lejanas donde las compañías tenían que visitar las páginas *Web* de los proveedores una a una para realizar los pedidos.

Ahora, se orienta la red a los empresarios para proporcionar una relación centrada en el cliente y ofrecerles toda la información que puedan necesitar. Con este explosivo cambio en la tecnología del *marketing* se pretende que los empleados utilicen potentes *PC's* para actualizar los datos constantemente y que *Internet* se convierta en una auténtica libertad de elección.

VERSIÓN DE IRIS FAILSAFE PARA APLICACIONES LINUX

SGI y *SuSE* han anunciado el inicio de un proyecto conjunto de ingeniería para llevar a cabo *IRIS Failsafe*. La ejecución de este programa permitirá al usuario *Linux* conectar dos o más servidores de tal manera que uno de ellos asuma toda la carga de trabajo en el caso de que el otro falle. Asimismo permitirá que las aplicaciones aumenten su nivel de disponibilidad en las operaciones críticas de los centros de datos. Tanto *SGI* como *SuSE* esperan tener el *software* disponible a mediados de este año.

Ambas compañías cuentan con un reconocido prestigio dentro del mundo de la informática. *SGI* es una compañía que proporciona un gran abanico de soluciones gráficas e informáticas avanzadas y tecnología de alta calidad, se ha centrado en los últimos años en la tecnología de código abierto durante varios años. En el caso de *SuSE*, es una compañía con amplio reconocimiento internacional dentro del mundo *Linux*. Cuenta con servicios cualificados de consultoría, soporte para el uso de *Linux* completo y ofrece *software* comercial.

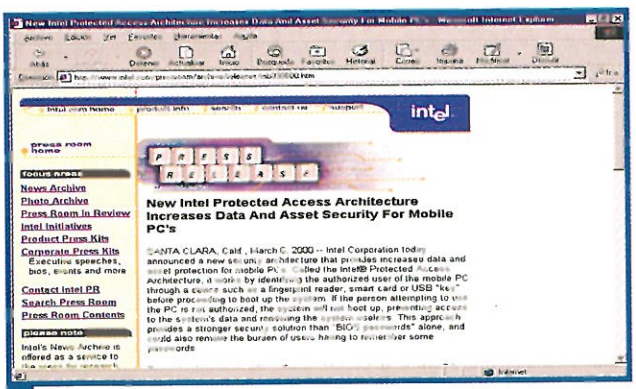


Página *Web* con el anuncio del acuerdo entre *SGI* y *SuSE*.

NUEVA ARQUITECTURA DE SEGURIDAD PARA PORTÁTILES

La *Intel Protected Access Architecture,* una tecnología que proporciona una mayor protección de datos y bienes a los ordenadores portátiles, ya está lista.

Esta arquitectura de seguridad desarrollada por *Intel Corporation* opera antes de que se cargue el sistema operativo. Funciona al identificar al usuario autorizado del ordenador portátil a través de una tarjeta inteligente o una "llave" *USB* antes de arrancar el sistema. Además se puede utilizar una contraseña o un código numérico para asegurar la identificación. Hay que tener en cuenta que la "llave" se almacena en un lugar protegido del ordenador y no en el propio disco duro. De esta manera se impide el robo de datos aun cuando se roba el disco duro.



Intel presenta la nueva seguridad de los ordenadores portátiles.

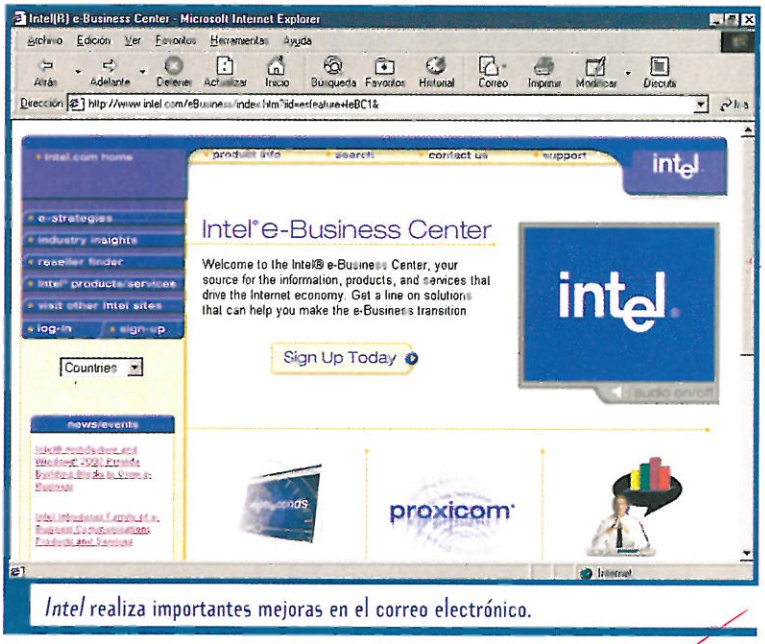
Tras la autenticación del usuario, la unidad del disco duro se desbloquea automáticamente y el ordenador arranca y carga el sistema operativo y las aplicaciones.

Según *Intel* esta propuesta es aún más segura que las "contraseñas de *BIOS*", y actúa como elemento disuasorio frente a los ladrones de sistemas, logrando que un ordenador portátil robado sea inútil.

Los ordenadores portátiles con esta nueva tecnología saldrán al mercado a principios del año 2001. Si tenéis más curiosidad sobre el tema podéis consultar la página: http://www.intel.com.

EBSP: UNA SOLUCIÓN PARA OPTIMIZAR EL COMERCIO ELECTRÓNICO

Intel lanza *e-Business Solution Provider* (*eBSP*). Un programa constituido para unir de manera rápida la *Web* con el comercio electrónico, orientado a compañías que están empezando a hacer negocio en *Internet* y que reclaman soluciones de comercio electrónico de extremo a extremo.



Intel realiza importantes mejoras en el correo electrónico.

Esta utilidad ofrecerá una alta disponibilidad en los recursos de la *Web*, tanto en productos como en herramientas, actividades de *marketing* y eventos organizados para dar a conocer a múltiples fabricantes el *software*, equipos y proveedores de servicios de *Internet*. Se podrá contar también con información técnica para mejorar las relaciones entre *e-Business* y sus clientes. Presenta además la posibilidad de ofrecer sus servicios en la página *Web e-Business Center* de *Intel*.

Los *eBSP* tendrán acceso a la base de datos de *Intel* de colaterales, artículos técnicos e información sobre soluciones, así como al *hardware* de cliente y servidor, *software* y productos de red de *Intel*. Los ingenieros de esta compañía esperan conseguir soluciones construidas en la plataforma de *Allaire* que puedan ser reproducidas con facilidad en las instalaciones de clientes.

PROCESADORES A 1 GIGAHERZIO

Ya están disponibles los procesadores a 1 *GHz* tanto de *AMD* como de *Intel*. En una carrera contra reloj estas dos compañías han anunciado la aparición de sus nuevos *chips* con sólo dos días de diferencia. Primero se anunció *Athlon* y después llegó el anuncio sobre *Pentium* III. Ambos estarán en el mercado español en el mes de junio.

Sin embargo del *Pentium* III ya sabemos que viene con un 15% más de prestaciones sobre los procesadores más rápidos para *PC's* del mundo. Así parecen corroborarlo los datos técnicos que ha mostrado *Intel* basados en los bancos de prueba estándares más usados en la industria.

El procesador *Pentium* III a 1 *GHz* ya está disponible en cantidades limitadas y lo incorporan fabricantes informáticos líderes. Dicho procesador en encapsulado *SECC2* tiene el precio de $990 en cantidades de 1.000 unidades. Las prestaciones son de primera categoría tanto para los sitios en *Internet* ricos en multimedia, como las aplicaciones avanzadas en fotografía digital, edición de vídeo, música, reconocimiento de voz y los juegos en *3D*. Aunque seguro que el *Athlon* a 1 *GHz* tampoco se queda atrás.

El nuevo procesador *Intel Pentium* III a 1 *GHz* llega al mercado.

ENTORNOS DE PROGRAMACIÓN

ENTORNOS DE PROGRAMACIÓN

C STYLED SCRIPT LANGUAGE (CSS) 3.01

Lenguaje de *scripts* con sintaxis similar al lenguaje *C*. Se integra en nuestro editor favorito y se ejecuta directamente como cualquier *script PERL, BAT, CMD* o *REXX*. Utiliza una sintaxis similar al *C* por lo que resulta conocido para programadores de *C, C++* o *Java*.

DZSOFT PERL EDITOR 2.0

Herramienta de programación de *CGI* en *Perl*. Escribe, edita y depura *scripts CGI* en *Perl* para *Windows* y *Unix*. Presenta una interfaz amigable e intuitiva, con accesos directos para numerosas funciones, que facilita el trabajo tanto a programadores avanzados como inexpertos.

IBM WEBSPHERE STUDIO 3.0.2 ENTRY

Agiliza la creación, gestión y depuración de aplicaciones *Web* multiplataforma. Se trata de una colección de herramientas muy fáciles de usar que incluyen asistentes para generar páginas basadas en bases de datos, actualización y corrección automática de enlaces, soporte para *JSP's, HTML, JavaScript, DHTML* y depuración de código (de forma local y remota).

SEVENSTEPS 2.0

Desarrolla sistemas profesionales de ayuda en línea. Incluye un potente editor, un método que asegura la rapidez y sencillez de manejo del proyecto. Puede guardar los ficheros en formato *HTML* o *WinHelp*.

THE GAMES FACTORY 1.5

Crea juegos de arcade, aventuras y salvapantallas. Permite presentar los objetos utilizando las funciones de animación o una secuencia externa *FLI* o un video-clip multimedia. Sus opciones permiten a los personajes recorrer distintos espacios, saltar obstáculos, etc.

VISUAL PERL EDITOR 2.5

Crea *scripts CGI* en *Perl*. El editor presenta dos paneles, uno para editar el *script* y otro para previsualizar el *CGI* en un navegador interno. También permite probar el *script* para detectar errores, o ejecutarlo normalmente.

LENGUAJE

VISUAL BASIC

BINARYWORK MP3 OCX 2.0.0

Control *ActiveX* (*OCX*) que reproduce ficheros de audio *MP3*. Puede obtener y mostrar información sobre los ficheros, crear rápidamente un reproductor y enviar el resultado a *Wavemapper, DirectSound* o a un fichero *PCM*. Incluye un proyecto *VB* de ejemplo.

MORFIT 3D ENGINE SDK 3.0

Genera apasionantes juegos *3D*. Es un motor de orientación general que no se limita a una categoría de juego, e incluye optimizaciones para juegos de interior y al aire libre.

PNFILEX 1.1.1

Componente *ActiveX* para el desarrollo de aplicaciones en *Visual Basic*. Incluye más de 60 funciones muy fáciles de usar que ofrecen información sobre ficheros comunes, del sistema, cadenas y tareas de manipulación.

JAVA

1ST JAVA NAVIGATOR 5.5

Incorpora a un sitio *Web*, una *intranet* o un sistema *extranet*, una selección de diferentes estilos de navegación. Es un programa complejo, pero fácil de usar y totalmente configurable; puede personalizar su apariencia utilizando los elementos que incorpora como temas, imágenes *2D* y *3D* y sonidos.

COCKTAIL STANDARD EDITION 1.1

Herramienta de generación de código *Java* que representa la estructura de los programas en forma de árbol. Con *Cocktail Standard Edition* el programador puede añadir nuevos componentes a las aplicaciones como si fuesen las hojas de un árbol y guardar la estructura en formato *XML*.

CODEGUIDE 2.0.2

Entorno inteligente de desarrollo en *Java* que facilita la depuración de los programas. Analiza las aplicaciones según se van desarrollando, muestra instantáneamente los errores y ofrece una potente ayuda en la programación, ya que no es necesario compilar primero para detectar los errores.

JQUICKCOMPILE 1.0

Rápido compilador para *Java* 1.2. Utiliza las clases estándar que proporciona *SUN* para analizar el código y generar los ficheros de clases. Consigue un mayor rendimiento, al

eliminar el tiempo de demora de iniciar el proceso y cargar las clases.

JSUITE 3.0

Colección de *JavaBeans* para mejorar la interfaz de usuario de las aplicaciones *Java*. Incluye 19 componentes entre los que podemos encontrar un calendario, una tabla de datos y una vista en forma de árbol.

KAWA 3.51A

Entorno de desarrollo integrado para aplicaciones *Java*. Proporciona un excelente entorno para desarrollar aplicaciones y aprender *Java* al mismo tiempo. Incluye un gestor visual de proyectos, un editor de clases y visores de clases y código fuente.

HTML

SITEDESIGNER 2.0.7

Editor visual *HTML* para la construcción de sitios *Web* profesionales. Incluye plantillas, *applets Java* y elementos que pueden ser utilizados para generar fondos, logotipos, etc.

SYNEDIT 2.4

Editor de texto que ofrece soporte para 30 lenguajes de programación. Cuenta con un visor de ficheros integrado, una ventana de fragmentos de texto para guardar las frases y palabras clave más utilizadas, una lista de *tags HTML* que facilita la introducción de sintaxis *HTML* en los trabajos y un editor binario/hexadecimal.

OTROS

3DCHARTING TOOLKIT 4.0

Conjunto de componentes *ActiveX* que facilitan la creación de presentaciones 2D y 3D. Incluye *3DChart, 3DPieChart, 3DLegend, 3DClock* y *3DText*. Permite generar gráficos de barras *2D* y *3D*, circulares, líneas, etc.

[illegible] SIMULATOR 1.2.1

[illegible]mblador y simulador para [illegible]oprocesadores 8085 basado en *Windows*. 8085 *Simulator* está pensado para personas que están aprendiendo a programar el microprocesador *Intel* 8085. Presenta un *IDE* integrado con funciones de depuración.

ALCATRAZ 3.0

Proporciona funciones de protección de copias a una aplicación. Es una librería estática de *C++* que permite crear y distribuir versiones de demostración (con limitaciones de tiempo y otros parámetros).

POWERTCP WEBSERVER TOOL 1.0

Crea aplicaciones para *Internet*. Trabaja junto con un control de servicio *NT* para agilizar la construcción de las aplicaciones. Permite reutilizar el código de cliente y servidor, publicar las aplicaciones y añadir una nueva interfaz de *Web*.

PRODELPHI 7.5

Reutilización y mantenimiento de código fuente para *Delphi*. Permite guardar los métodos de un programa, insertar expresiones al inicio y final de un procedimiento, optimizar y depurar el código fuente de programas *Delphi*.

HERRAMIENTAS Y UTILIDADES

CLICKTEAM INSTALL MAKER PRO 1.3

Herramienta de distribución de programas. Permite tener todo almacenado en un fichero ejecutable para su distribución, que sigue los estándares de los programas de instalación.

CODE SNIP 2.0

Base de datos para guardar fragmentos de código. Permite guardar el nombre del programa, la fecha, el fragmento del código y una descripción de lo que hace. *Code Snip* es realmente una colección de múltiples bases de datos en una aplicación.

DOCUCLASS 1.0

Genera documentación del código fuente en *C++*. Crea documentos de referencia de clases de apariencia profesional, para librerías *C++*, aplicaciones y proyectos. Puede generar y actualizar automáticamente la documentación desde el código fuente y los comentarios.

HELPBREEZE JAVAHELP EDITION 1.0

Herramienta de autor diseñada específicamente para crear sistemas *JavaHelp*. *JavaHelp* es el formato estándar de ayuda en línea creado por *SUN*, para su uso en *applets* y aplicaciones *Java*.

INSTALL-US PRO 3.7

Conjunto de herramientas para crear programas de instalación para profesionales. Incluye un asistente para crear sencillos programas de instalación en pocos minutos, pero además ofrece la posibilidad de definir cualquier parámetro de instalación.

RESOURCE-GRABBER 2.50

Localiza y extrae *bitmaps*, imágenes de botón, iconos, cursores y sonidos que se encuentran en el ordenador. Explora programas, librerías y otro tipo de ficheros para obtener elementos multimedia que puedan ser insertados en nuevos programas.

REDES

602PRO LAN SUITE LITE 2000.0.1.24

Proporciona acceso a *Internet* a tres usuarios conectados en red. Estos usuarios pueden utilizar la misma cuenta y conexión, proteger sus equipos con un *firewall* interno, hacer un seguimiento de la actividad, realizar conversiones entre fax y correo electrónico y usar cualquier cliente de correo *POP3* y los navegadores de *Internet* más populares.

ATELIER WEB TCP PORT SCANNER 1.20
Escanea los puertos de una red. Puede obtener de forma remota información del *host* como el nombre y sus alias. Esta actualización ofrece mayor rapidez, una base de datos de puertos más amplia, una opción para establecer el número de puertos que se desean escanear y mejoras en la interfaz de usuario.

DOMAIN NAME ANALYZER 1.5B
Agiliza la búsqueda de nombres de dominio para productos y servicios. Primero, proporciona múltiples nombres que genera a partir de algunas palabras clave y luego comprueba si se encuentran disponibles.

NETINFO 3.7
Colección de herramientas de diagnóstico de redes. Muy útil para usuarios de *Internet* e *intranets*, ofrece servicios como monitorización *IP*, búsqueda de nombres de dominio, seguimiento de las conexiones, del estado del servidor, etc. Entre otras utilidades incluye *Local Info*, *Ping*, *Trace*, *Look Up*, *Finger*, *Whois*, *Daytime*...

WHATSUP GOLD 5.0
Utilidad de monitorización del estado de redes. Es capaz de hacer un seguimiento de grandes redes jerárquicas y mostrar estadísticas de las redes desde un servidor *Web*.

WINROUTE PRO 4.1A
Conecta una red a *Internet* utilizando una única dirección *IP*. *WinRoute Pro* es un *firewall*, un *router*, un servidor *proxy* de caché y un servidor de correo. Soporta las aplicaciones y protocolos *HTTP, HTTPS, Gopher, FTP, POP3, SMTP, Telnet, RealAudio, DNS, Finger, Talk, ICQ, H.323, PPTP, IPX* sobre *IP, ICA WinFrame, IRC, SSH, SNMP, TFTP, ICMP, Ping, Traceroute, Whois-UUCP* sobre *IP* y *Whois*.

WINYXY PROXY SUITE 1.21
Firewall y *proxy* que soporta *DNS, FTP, HTTP, HTTPS, PNA, SOCKS* 4 y 5, y protocolos *TelNet*. Permite establecer restricciones de entrada y salida para cada servicio, compartir una misma conexión y hacer un seguimiento de todos los servicios.

ZONEALARM 2.0.26
Detecta los intentos de acceso al ordenador a través de *Internet*. Es una utilidad de seguridad y *firewall* que ofrece varios servicios de protección, para las comunicaciones y trabajo con *Internet*.

DOCUMENTACIÓN/ TUTORIALES

INTRO TO HTML 1.1
Manual de formación sobre los elementos de creación de páginas *Web* con *HTML*. Abarca temas como el formato de texto, tablas, formularios, listas, imágenes, hojas de estilo, animaciones, *JavaScripts*, objetos *ActiveX*, *VBScript*, mapas de imágenes, etc.

MAZEBOTS 1.00
Programa de simulación robótica. Utiliza un lenguaje potente pero amigable, por lo que puede ser utilizado desde para aprender programación hasta para probar sistemas de inteligencia artificial. Permite diseñar robots y programarlos para que atraviesen laberintos; los robots incluso pueden "recordar" información.

WEBTUTOR 3.9
Excelente conjunto de tutoriales sobre creación de páginas *Web* con *HTML*. Han sido creados pensando tanto en autores principiantes como avanzados. Por ello, están ordenados según las explicaciones se refieran a los principios de *HTML*, o a temas de mayor complejidad como la creación de elementos avanzados (tablas, formularios...).

ATENCIÓN:
En caso de problemas con el CD-ROM envíelo por correo ordinario, a la atención del SERVICIO TÉCNICO DE SÓLO P., incluyendo en el interior del sobre sus datos personales, a la siguiente dirección:
C/ San Sotero, Nº 5, 1ª Planta, 28037 (Madrid)

IMPRESCINDIBLES

ANTIVIRUS
AntiViral Toolkit Pro 3.0.129
AVP Virus Enciclopedia
eSafe Desktop 2.2.24
McAfee VirusScan 4.0.3
Panda Antivirus 6.15.00 Platinum
Quick Heal 5.23

GRÁFICOS
Icon Bank 4.0 Gold Edition
IconForge 4.6
MicroAngelo 98 v4.77
Paint Shop Pro with Animation Shop 6.02
Reptile 2.01
SureThing CD Labeler 2.0
ThumbsPlus 4.02
Ultra Fractal 2.04
Xara WebStyle 1.2

INTERNET
AutoWinNet 6.0 Beta 1
Copernic 2000 v4.1
Cuentapasos 3.77
CuteFTP 3.5.6
Dial-Up Magic 1.8
Eudora Light 3.0.6
Free Agent 1.21
GetRight 4.1.2
Go!Zilla 3.5
Guardián 1.1
HotDog Professional 5.5
ICQ 99b beta 3.19 build 2569 Rev.A
i.Share 3.5
Jet Color 1.20
Mass Downloader 1.2.87
MIRC 32 5.7
SubmitWolf ES 4.02.002 (Añadir Pro)
TS-ImageMapper 2.00
URL Organizer 2.3.2
WebZIP 3.07
WinGate 3.0.5
XMLwriter 1.1

MULTIMEDIA
AudioCatalyst 2.01
COWON Jet-Audio 4.7
MusicMatch Jukebox 5.0
Sonique 1.30
WinAMP 2.61

NAVEGADORES
Internet Explorer 5.01
NeoPlanet 5.1
Netscape Communicator 4.72
Opera 3.61

PROGRAMACIÓN
Decafe Pro 3.6
Free Pascal v0.99.14
Hackman 4.02
Help & Manual 2.50
InstallConstruct 3.3
Java Development Kit 1.2.2-001
UltraEdit Professional Text/HEX Editor 7.00b
Windows Registry Guide 1.3
WinHex 9.26

UTILIDADES
Adobe AcrobatReader 4.0
Advanced Registry Tracer (RegFix) 1.20
Babylon Translator Spanish 2.2
CallCenter 3.5.8
DirectX 7.0a
Emergency Recovery System 9.04
Nero 4.0.8.3
SiSoft Sandra 2000.3.6.4
WebTrends Professional Suite 4.0b
Where Is It? 2.16
Windows Commander 4.03
WinZip 7.0

Curso de Programación con WAP (I)

Constantino Sánchez Ballesteros.
Técnico Superior Desarrollo Aplicaciones Informáticas.

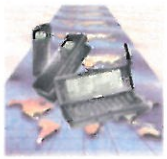
Comenzamos un nuevo tema que ahora está en la boca de casi todos los programadores de páginas *Web;* se trata de *WAP.* Con este soporte utilizaremos el lenguaje *WML* para crear páginas destinadas a los móviles de próxima generación, que ya son una realidad, y que permiten visualizar contenido interactivo publicado en *Internet.*

INTRODUCCIÓN

Los fabricantes de *hardware* (en ellos se incluyen creadores de móviles, *PDA's,* etc.) han comenzado a desarrollar dispositivos capaces de ver contenido publicado en *Internet* a través de ellos. Para que esto sea posible, se ha tenido que crear un lenguaje *(WML)* adaptado a las características especiales de estos dispositivos.

Si pensamos un poco, es muy difícil actualmente que se puedan visionar páginas *Web* de *Internet* a través de móviles (imágenes en alta resolución, sonidos, *XML,* controles *ActiveX,* etc.) debido al volumen de datos que se debe recibir, a las características del *hardware,* etc. En este sentido, el contenido que se puede mostrar mediante el uso de la tecnología *WAP* está estrechamente relacionado con las capacidades de los dispositivos utilizados (móviles, *PDA's,* etc.), teniendo muy en cuenta sus limitaciones tanto técnicas como físicas.

Figura 1.- Imagen de un móvil con soporte *WAP.*



Por supuesto, es posible que dentro de un par de años veamos cualquier página *Web* en un móvil, de igual modo que si lo hiciéramos a través de nuestro *PC*. Hay noticias por parte de *Microsoft* para visualizar páginas *HTML* en los móviles, aunque esto todavía está por llegar.

En esta serie de artículos vamos a basar la creación de contenido *WML* (lenguaje utilizado para los dispositivos *WAP)* para *WAP* desde el punto de vista de los móviles. El lenguaje *WML* se ha creado para el desarrollo de páginas sobre móviles y otros dispositivos de reducido tamaño y *hardware* limitado, de igual modo que se utiliza *HTML* para los ordenadores.

WAP es el soporte Internet para los móviles

Antes de comenzar debemos tener en cuenta varios aspectos. Para programar páginas *WAP* es necesario un entorno de programación que permita emular las páginas creadas en un móvil virtual. De este modo podremos ver *in situ* cómo quedaría la página, al igual que si tuviéramos el móvil en la mano y visualizáramos su pantalla.

Existen varias empresas que ya han creado sus herramientas de programación para crear contenido para *WAP.* Entre estas compañías cabe destacar:

- *Nokia*
- *Ericsson*
- *UpPhone*

Dada la facilidad de uso que presenta el *software* de *Nokia,* nos basaremos en éste para realizar nuestros programas en *WML.* Por supuesto, si el lector considera cualquier otra opción, podrá crear contenido para *WAP* sin ningún problema, siempre que el entorno utilizado soporte las especificaciones implementadas para *WAP* y *WML.*

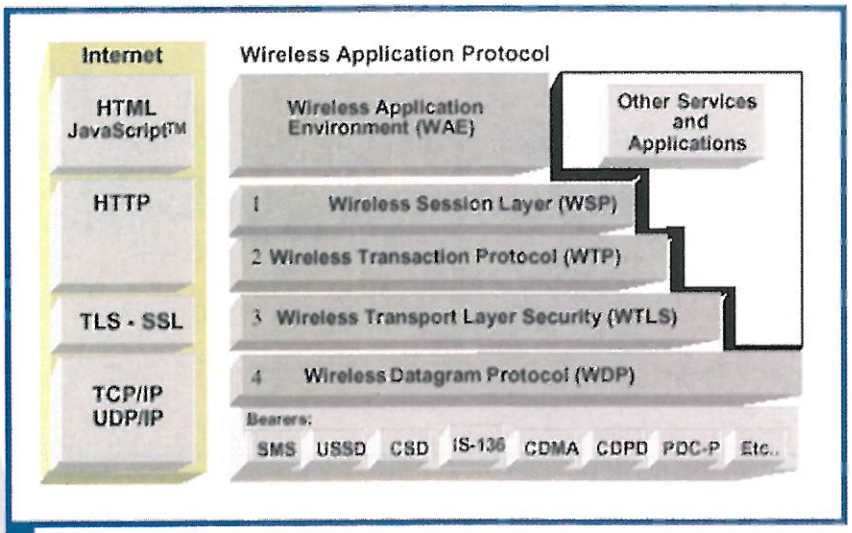


Figura 2.- Protocolo Wap.

NOKIA WAP TOOLKIT

El *Nokia WAP Toolkit* permite a los desarrolladores un entorno para crear, probar y visualizar las aplicaciones *WAP*. El *software* incluido con el *Nokia WAP Toolkit* es el siguiente:

- Codificadores de *WML* y *WMLScript.*
- Cliente para simulación *WAP* con un móvil genérico para interfaz de usuario.
- Aplicaciones *WAP* con soporte de depurado.
- Documentación de usuario.

El *Nokia WAP Toolkit* simula un teléfono móvil *WAP* genérico, que denominaremos agente de usuario. Un agente de usuario *WAP* es similar a un navegador de *Internet* como *Netscape,* con la distinción de que las *URL's* se solicitan sobre redes celulares convencionales. La solicitud de *URL* es enviada a un servidor que contiene servicios *WML (Wireless Markup Language)* y *WMLScript (Wireless Markup Language Script).* Estos servicios responden a las peticiones utilizando a su vez *WML* y *WMLScript.*

WML y WMLScript son los lenguajes utilizados para visualizar páginas Web en móviles

WML se basa en *XML* y fue desarrollado principalmente por el *Wireless Application Protocol (WAP) Forum* para la especificación de contenido.

Dentro de las herramientas del *Nokia WAP Toolkit* se incluyen los siguientes componentes:

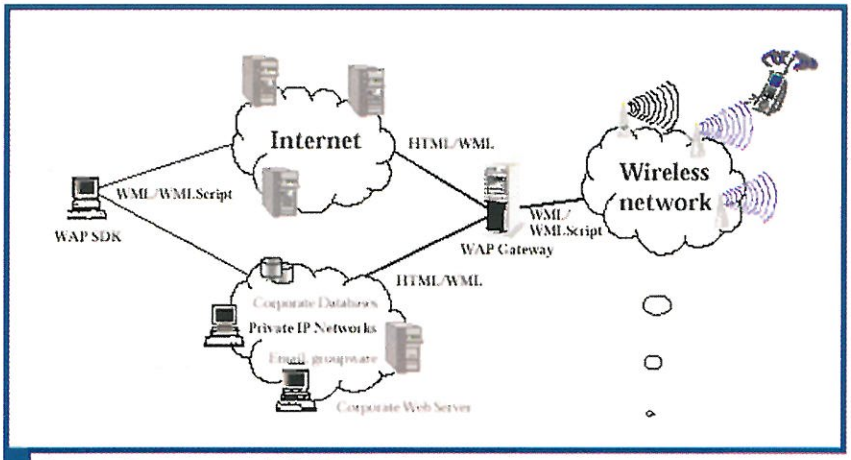


Figura 3.- Funcionamiento de *WAP*.

- *Nokia WAP Toolkit application* para *Windows NT 4.0, Windows 95*, y *Windows 98*, que interpreta *WML* y *WMLScript* permitiéndonos crear, editar y depurar servicios *WML* y *WMLScript*. El cliente para la simulación incluye a su vez los siguientes componentes:

  1. *WML browser*, con intérprete *WMLScript* y sus librerías correspondientes.
  2. Móvil *WAP* para simulación.
  3. Codificadores para *WML* y *WMLScript*.
  4. Editores *WML*, *WMLScript* y *WBMP*.
  5. Protocolo *WAP*, *HTTP* y módulos de acceso a archivos.
  6. Vistas para depurado *(Debugger)*.

- Programas de ejemplo con código fuente y programa de instalación.

- Librería *XML* que podemos utilizar para construir documentos *WML* válidos para su utilización por los componentes del servidor o para uso general. La librería sólo nos permite crear contenido válido, validado por el estándar *DTD* para *WML1.1*.

- Documentación de usuario en formato *PDF (Adobe Acrobat)*.

Los componentes del *Toolkit* de *Nokia* se pueden dividir en tres categorías que se detallan en los siguientes apartados.

SIMULADOR

El simulador lee contenido *WML* o *WMLScript* y muestra el resultado en la ventana de simulación de la interfaz de usuario, que se parece a un teléfono móvil.

EDITORES

Como ya se ha dicho anteriomente, las aplicaciones *WAP* se crean en *WML* y *WMLScript*. El *Toolkit* de *Nokia* incluye un editor de texto para escribir el código que creamos oportuno. Los datos de nuestras aplicaciones se cargarán a partir de ficheros locales.

DEPURADOR (DEBUGGER)

También se incluye una herramienta que visualiza información útil como valores de variables.

REQUERIMIENTOS

Para poder ejecutar el *Toolkit* de *Nokia* necesitaremos las siguientes características *software* y *hardware* mínimas:

- *Windows NT 4.0* con el *Service Pack 3* instalado, *Windows 98* o *Windows 2000*.
- *Pentium* 266 *MHz*.
- 20 *Mb* de espacio en disco duro.
- 64 *Mb* de *RAM*.
- Tarjeta gráfica con soporte de resolución a 65.000 colores.
- *Java™ Runtime Environment* 1.2.2 o superior. Este programa podemos bajarlo de la dirección **www.javasoft.com/products/jdk/1.2/jre/index.html** de la *Web* de *Sun Microsystems*.

CONCEPTOS SOBRE WAP

Una aplicación para *WAP* consiste en una aplicación servidora y una aplicación cliente. Mediante un *gateway* se descarga el contenido al agente de usuario (móvil) para su ejecución. Por supuesto, se necesita un entorno estándar para que la misma aplicación cliente se ejecute sobre diferentes móviles.

El navegador utilizado para *WAP* es muy similar a un navegador de páginas *Web*. La diferencia principal es que alberga contenido *WML*. El navegador también tiene un intérprete de *scripts* para ejecutar aplicaciones en el agente de usuario (móvil).

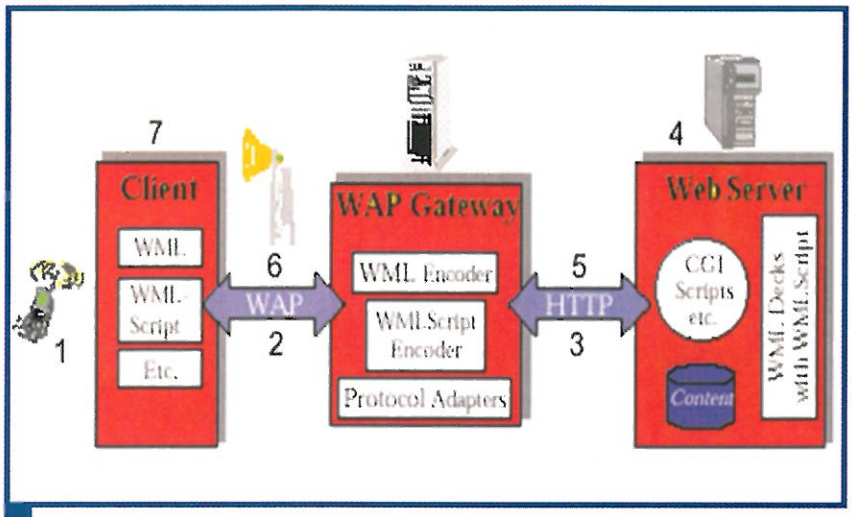


Figura 4.- Arquitectura del protocolo de *Wap*.

Estas aplicaciones están escritas en un lenguaje de tipo *script* denominado *WMLScript*. Además del lenguaje de programación en sí, el intérprete de *scripts* también implementa un conjunto de librerías que permiten a las aplicaciones el acceso a ciertos servicios del agente de usuario.

WML y *WMLScript* se han diseñado específicamente para redes sin cable, y se codifican en binario para una eficiencia de transmisión óptima.

El protocolo de *WAP* tiene cuatro capas:

- Sesión.
- Transacción.
- Seguridad.
- Datagramas.

PROTOCOLO WAP

En algunos casos, la aplicación o cualquier otro contenido se localiza en un servidor *Web*. Este contenido puede ser nativo de *WAP*, creado con *WML* y *WMLScript*, o puede ser *HTML*. Algunos *gateways* son capaces de traducir documentos *HTML* a *WML* directamente. En *WAP*, el contenido y las aplicaciones son direccionadas con una *URL*, del mismo modo que otros protocolos de *Internet*.

WAP es similar al modelo *Web*, trabajando del modo siguiente:

BUSCAMOS COLABORADORES

Si además de leer

SÓLO PROGRAMADORES

te gusta la programación, y quieres escribir
en tu revista, no dudes en ponerte en contacto
con nosotros. Envíanos tu propuesta junto a tu curriculum
a la siguiente dirección:

REVISTAS PROFESIONALES
C/San Sotero, 5 - 1.ª planta
28037 Madrid
Ref.: Colaboraciones Sólo Programadores
E-mail: solop@virtualsw.es

TABLA 1. La función Random de la librería Lang, al detalle.

LIBRERÍA *LANG*

Esta librería contiene un conjunto de funciones propias del lenguaje *WMLScript*.

RANDOM

Función: random(value)

Descripción: Devuelve un valor entero positivo mayor o igual a cero pero menor o igual al valor dado. El valor devuelto se elege de manera aleatoria. Si value está en coma flotante, tendremos que utilizar primero Float.int( ) para efectuar la conversión a valor entero.

Parámetros: *value=Integer*

Valor de retorno: entero o inválido.

Excepciones: si *value* es igual a cero, la función retorna cero. Si es menor que cero la función retornará inválida.

Ejemplo:

```
var a=10;
var b=Lang.random(5.1)*a; // b=0..50
var c=Lang.random("string"); // c=inválido
```

1. El usuario presiona un botón del móvil que tiene asignado una petición de *URL*.
2. El móvil envía la petición a un *WAP* gateway utilizando el protocolo *WAP*.
3. El gateway crea una petición *HTTP* convencional para el *URL* especificado y la envía al servidor *Web*.
4. La petición *HTTP* es procesada por el servidor *Web*. La *URL* se puede referir a un fichero estático, un *CGI* o cualquier aplicación de tipo *script*.
5. El servidor *Web* devuelve un *Deck WML* (más adelante veremos qué es un *Deck)* con la cabecera *HTTP* o la salida *WML* del *CGI* o la aplicación *script*.

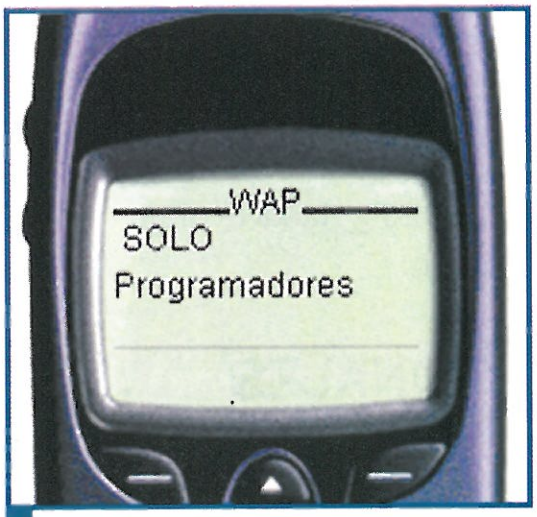


Figura 5.- Imagen con el primer ejemplo del artículo.

6. El *WAP gateway* verifica la cabecera *HTTP* y el contenido *WML* codificándolo a formato binario. Llegado a este punto, el *gateway* crea una respuesta *WAP* conteniendo el *WML* y lo envía al móvil que hizo la llamada.
7. El móvil recibe la respuesta *WAP* visualizando el primer *Card* del *Deck WML*.

En un apartado posterior veremos los *Decks* y *Cards*, que son la columna vertebral de las aplicaciones *WML*.

WML

El *Wireless Markup Language (WML)* es un lenguaje que describe cómo presentar contenido *WAP* al usuario. Con *WML* podemos visualizar información sobre un teléfono móvil, permitir al usuario opciones de entrada de datos, y especificar cómo el agente de usuario debe responder cuando el usuario introduce opciones, activa una función o presiona un botón.

DECK Y CARD

La unidad básica del *WML* es un *Card*, que especifica una simple interacción entre el móvil y el usuario. Los *Cards* se agrupan conjuntamente en *Decks*. Un *Deck* es el elemento de más alto nivel de un documento *WML*. Cuando un móvil recibe un *Deck*, activa el primer *Card* del *Deck* a menos que se haya redireccionado a un *Card* diferente. Se trata de una analogía entre una baraja y las cartas que la componen.

Una página WML se compone principalmente de un Deck y Cards

Vamos a ver nuestro primer ejemplo en *WML* utilizando un *Deck* con un simple *Card:*

```
<?xml version="1.0"?>
<!DOCTYPE wml PUBLIC „-
    //WAPFORUM//DTD WML 1.1//EN"
„http://www.wapforum.org/DTD/wml_1
    .1.xml">
<wml>
```

LISTADO 1

```
<?xml version="1.0"?>
<!DOCTYPE wml PUBLIC "-//WAPFORUM//DTD WML 1.1//EN"
"http://www.wapforum.org/DTD/wml_1.1.xml">

<wml>
 <template>
   <do type="prev" label="atras">
      <go href="#card1"/>
   </do>
 </template>
      <card id="card1" ontimer="#card2" title="TIMER">
          <timer value="30" />
    <p>
    PRUEBA DE TIMER!!!
  </p>
  <do type="ACCEPT" label="SIGUIENTE">
                <go href="#card2" />
  </do>
  </card>
  <card id="card2" title="INTRODUCIR" newcontext="true">
  <p>
  NOMBRE: <input format="*M" name="S_NOMBRE"
                      title="NOMBRE:" value=""/>
  <do type="prev" label="continuar">
      <go href="#card3" />
  </do>
  </p>
  </card>
  <card id="card3" title="URLs">
      <p>
       <small>
       Hola $(S_NOMBRE) <br/>
       Tus links... <br/>
       </small>
       -<a href="http://www.solopro.wml" title="Mi revista ."
/> <br/>
       -<a href="http://www.yahoo.wml" title="buscador ." />
<br/>
     </p>
  </card>
</wml>
```

```
<card id="primero" title="WAP">

<p>
Solo
Programadores
</p>

</card>

</wml>
```

1. Las tres primeras líneas (realmente es una línea de código) representan una instrucción estándar *XML* requerida por todos los documentos *XML*. De hecho, cuando indiquemos al entorno de desarrollo de *Nokia* que queremos hacer una página nueva aparecerá automáticamente.

2. La siguiente línea es la declaración del tipo de documento *XML*, también requerida por todos los documentos *XML* que utilicen tipos de documento externo, tal y como lo hace *WML*.

3. La tercera línea es el elemento raíz del documento *XML*, que debe ser único y debe corresponder al nombre del elemento raíz indicado en el primer argumento del tipo de documento. Todos los documentos *WML* deben comenzar con un *tag* <*wml*> y finalizar con otro *tag* </*wml*>.

4. Las siguientes líneas definen un *Card* que contiene un *tag* de comienzo y uno de fin, incluyendo el texto que será visualizado por el usuario. Para el *tag* de comienzo es necesario indicar su identificador *(ID)* y el título utilizado para la página a visualizar.

Como ya se ha visto, *WAP* también incluye *WMLScript* además de *WML*, lenguaje tipo *script* que podemos utilizar junto con *WML* para incluir inteligencia a los servicios de cliente. *WMLScript* también puede utilizarse como herramienta independiente. El lenguaje se basa en *ECMAScript*, aunque ha sido modificado para tener un soporte mejorado en dispositivos con bajo ancho de banda como los teléfonos móviles.

WMLScript permite realizar tareas no implementadas en WML

Con *WMLScript* podemos "pisar" las siguientes restricciones de *WML:*

- Chequear la validación de la entrada de datos por parte del usuario.

- Facilidades de acceso para el móvil. Por ejemplo, sobre el móvil, se permite al programador realizar llamadas por teléfono, enviar mensajes, agregar números de teléfono a la libreta de direcciones o acceder a la tarjeta *SIM*.

- Generar mensajes y diálogos de forma local, permitiendo alertas, mensajes de error, confirmaciones, etc., que se verán más rápidamente por parte del usuario.

EJEMPLO DE WML Y WMLSCRIPT

El siguiente ejemplo que vamos a realizar será una mezcla de *WML* y *WMLScript*. El propósito del programa es generar un número aleatorio y visualizar el resultado:

Figura 6.- Primer *card* en ejecución.

TABLA 2. Direcciones de interés.

//Para descargar el Toolkit
www.forum.nokia.com

//Para descargar el Runtime de java
www.javasoft.com/products/jdk/1.2/jre/index.html

//Página Web de Ericsson
www.ericsson.com

- El primer *Card (card1)* llama al documento *WML script* **ejemplo2.wmls** y pide a la función aleatorio que genere un número aleatorio.

- El segundo *Card (card2)* muestra el resultado de la operación *WMLScript*.

- El *script* **ejemplo2.wmls** genera un número aleatorio y devuelve el resultado al segundo *Card* del *Deck WML*.

```
<?xml version="1.0"?>
<!DOCTYPE wml PUBLIC "-//
    WAPFORUM//DTD WML 1.1//EN"
    "http://www.wapforum.org/DTD/
    wml_1.1.xml">

<wml>

<card id="card1" title="WML/
    WMLSCRIPT">

<p align="center">
NUM. ALEATORIO
(C) Constantino
</p>
<do type="ACCEPT" label=
    "ALEATORIO">
<go href="EJEMPLO2.wmls#
    aleatorio()"/>
</do>

</card>
<card id="card2" title=
    "RESULTADO">
<p>
Generado el: $(RESULT)
</p>
</card>

</wml>
```

La variable **RESULT** mostrará el resultado del código ejecutado en el archivo *WMLScript*. Para visualizar su contenido debemos anteponer a dicha variable el símbolo **$** y acotar el nombre con paréntesis ().

Tal y como se puede apreciar en el código, existen algunas similitudes entre el lenguaje *HTML* y *WML*.

Las Figuras 6, 7 y 8 muestran las pantallas visualizadas al ejecutar las tres partes de que consta el programa.

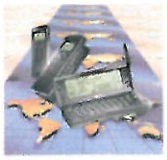
SUBPROGRAMA EN WMLSCRIPT

El módulo en *WMLScript* se encarga de generar el número aleatorio. A continuación se detallan todos los procesos del código:

- Genera un número aleatorio entre **0** y **100**. Este número será guardado en la variable **r**.

- Asignamos a la variable **RESULT** el contenido de **r**.

- Enviamos el resultado al **Card2** del fichero **ejem-**



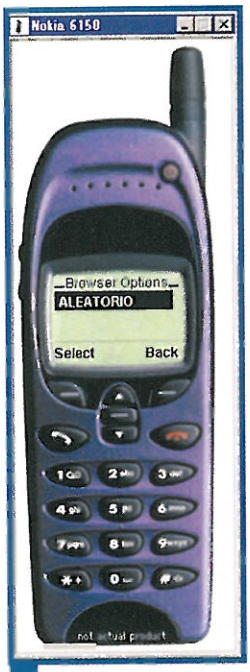


Figura 7.- Segundo *card* en ejecución.

plo2.wml y actualizamos los valores de las variables.

```
extern function aleatorio()
{
var r = Lang.random(100);
WMLBrowser.setVar("RESULT", r);
WMLBrowser.go("EJEMPLO2.wml#card2"
    );
}
```

Para asignar a una variable global el resultado se utiliza *WMLBrowser.setVar.* El otro método *(go)* permite volver a cualquier *Card* de una página que indiquemos entre los paréntesis. Con este sistema podemos crear bucles.

La sintaxis para definir la función es muy parecida al lenguaje *C*. Se utilizan llaves {} para definir un segmento de código y se utiliza **extern** para definir la función como externa, permitiendo referenciarla en otro archivo.

Para finalizar con el ejemplo, mediante **Lang.random** generamos el número al azar (0 a 100). Se trata de una librería que contiene algunas funciones útiles que iremos viendo a lo largo de la serie. En la Tabla 1 se muestra al detalle la función *Random* de la librería *Lang.*

TIMER

En el siguiente ejemplo vamos a utilizar un *Timer.* El programa utilizará tres *cards* (ver Listado 1).

- El primero mostrará un mensaje durante 3 segundos.
- El segundo *Card* pedirá introducir datos al usuario.
- El tercer *Card* mostrará varias *URL* para que el usuario pueda acceder a ellas con el móvil.

Para poder definir un tiempo en el *Timer* debemos establecer el valor en **TimerValue.**

El código WMLScript debe crearse en archivos diferentes a los de WML

Vamos a dar un repaso a las partes más reseñables del código. Mediante *Timer value* establecemos en segundos el tiempo de espera para mostrar el siguiente *Card.* Para poder ir de un *Card* a otro utilizaremos la sintaxis siguiente:

```
<go href="#cardX" />
```

Donde **X** representa el número del *Card* al que queremos dirigirnos.

Veamos la siguiente instrucción:

```
<do type="ACCEPT" label=
    "SIGUIENTE">
```

Con ella logramos que cuando en el móvil se muestre la entrada *ACCEPT,* ésta se sustituya por la palabra **SIGUIENTE.**

Mediante **<input format="*M" name="S_NOMBRE" title="NOMBRE:" value=""/>** permitimos la entrada de datos por parte del usuario. También definimos una variable *(S_NOMBRE)* que posteriormente nos permitirá visualizar en otra pantalla el nombre introducido por el usuario, de tal manera que tendremos amplias posibilidades de personalización de las páginas en función del usuario que haya entrado.

Finalmente, para visualizar los diversos enlaces a páginas que hayamos incluido utilizaremos la siguiente sintaxis:

```
-<a href="página_web"
    title="título visualizado en
    la página" />
```

En el próximo número veremos nuevas implementaciones, incluyendo datos dinámicos como los utilizados hasta ahora y la inserción de imágenes, para su utilización en *WAP.*

Figura 8.- Tercer *card* en ejecución.

SUSCRÍBETE A SÓLO PROGRAMADORES

LA PRIMERA REVISTA DE PROGRAMACIÓN EN CASTELLANO

QUE NO TE LÍEN CON EL <CÓDIGO>

- NOTICIAS
- MULTIMEDIA
- TEORÍA
- INTERNET
- DESARROLLO DE APLICACIONES
- COMUNICACIONES
- HERRAMIENTAS
- PROGRAMACIÓN
- ÚLTIMAS TENDENCIAS
- Y MUCHO MÁS

BOLETÍN DE SUSCRIPCIÓN

Rellene o fotocopie el cupón y envíelo a REVISTAS PROFESIONALES, S.L. (Revista Sólo Programadores).
C/ San Sotero, 5. 1ª Planta. 28037 Madrid. Tlf: 91 304 87 64. Fax: 91 327 13 03

Quiero suscribirme a la revista SÓLO PROGRAMADORES desde el Número .....
y beneficiarme de las condiciones de esta magnífica promoción:

❑ SUSCRIPCIÓN ANUAL
11 NÚMEROS + 11 CD-ROMs
AL PRECIO DE
8.580 ptas. / 51,57€

Precio de suscripción para el extranjero:
11 NÚMEROS + 11 CD-ROMs
AL PRECIO DE
11.800 Ptas.

FORMAS DE PAGO:

❑ Giro postal a nombre de REVISTAS PROFESIONALES, S.L
❑ Transferencia al Banco Popular Español. C/ Valdecanillas, 41.
Nº c/c: 0075/1040/43/ 0600047439
❑ Talón bancario a nombre de REVISTAS PROFESIONALES, S.L
❑ Domiciliación bancaria
❑ Contra reembolso

Soy antiguo suscriptor
❑ Sí ❑ No

PARA ENVÍOS AL EXTRANJERO SÓLO SE ADMITIRÁN LAS SIGUIENTES FORMAS DE PAGO:
❑ Giro postal a nombre de REVISTAS PROFESIONALES, S.L
❑ Eurocheque conformado con un banco español a nombre de REVISTAS PROFESIONALES, S.L
❑ Por VISA ____/____/____/____
Caducidad: __/__

NOMBRE Y APELLIDOS:.....
EDAD:..... PROFESIÓN: .....
TFNO: ..... DOMICILIO: .....
CIUDAD:..... C.P: .....PROVINCIA:.....

Datos de domiciliación:
Banco.....
Domicilio .....
Nº de Cuenta.....
Titular.....
Fecha .....
FIRMA

Promoción válida hasta agotar existencias

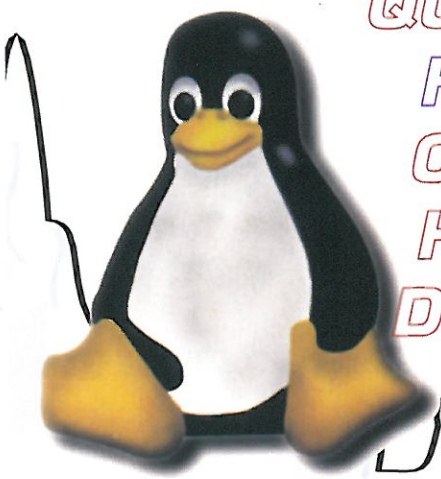
SUSCRIPCIÓN DOBLE

LA PRIMERA REVISTA DE PROGRAMACIÓN EN CASTELLANO

SÓLO PROGRAMADORES

SÓLO PROGRAMADORES LINUX

QUE NO TE LÍEN CON EL <CÓDIGO>



PARA NO QUEDARSE HELADO CUANDO HABLEN DE LINUX

BOLETÍN DE SUSCRIPCIÓN

Rellene o fotocopie el cupón y envíelo a REVISTAS PROFESIONALES, S.L. (Revista Sólo Programadores).
C/ San Sotero, 5. 1ª Planta. 28037 Madrid. Tlf: 91 304 87 64. Fax: 91 327 13 03

Quiero suscribirme a la revistas SÓLO PROGRAMADORES y SÓLO PROGRAMADORES LINUX desde el Número .....
y beneficiarme de las condiciones de estas magníficas promociones:

❑ **SUSCRIPCIÓN ANUAL**
22 NÚMEROS + 22 CD-ROMs
AL PRECIO DE
14.822 ptas. / 89,09€

Precio de suscripción para el extranjero:
22 NÚMEROS + 22 CD-ROMs
AL PRECIO DE
21.200 Ptas.

FORMAS DE PAGO:

❑ Giro postal a nombre de REVISTAS PROFESIONALES, S.L
❑ Transferencia al Banco Popular Español. C/ Valdecanillas, 41.
Nº c/c: 0075/1040/43/ 0600047439
❑ Talón bancario a nombre de REVISTAS PROFESIONALES, S.L
❑ Domiciliación bancaria
❑ Contra reembolso

Soy antiguo suscriptor
❑ **Sí** ❑ **No**

PARA ENVÍOS AL EXTRANJERO SÓLO SE ADMITIRÁN LAS SIGUIENTES FORMAS DE PAGO:
❑ Giro postal a nombre de: REVISTAS PROFESIONALES, S.L
❑ Talón conformado en dólares a nombre de: REVISTAS PROFESIONALES, S.L
❑ Por VISA ____/____/____/____
Caducidad: __/__

NOMBRE Y APELLIDOS:.....
EDAD:..... PROFESIÓN: .....
TFNO: ..... DOMICILIO: .....
CIUDAD:..... C.P: .....PROVINCIA:.....

Datos de domiciliación:
Banco.....
Domicilio .....
Nº de Cuenta.....
Titular.....
Fecha .....
FIRMA

Promoción válida hasta agotar existencias

Firmado de Applets (y II)

Applets

Javier Sanz Alamillo. *Ingeniero de Software.*

El firmado de *applets* permite que éstos puedan realizar las mismas tareas que una aplicación *Java* tradicional, posibilitando además independencia de instalación, es decir, se pueden ejecutar desde cualquier lugar en el que se disponga de un navegador y una conexión de red.

FIRMANDO DE APPLETS PARA INTERNET EXPLORER

Una vez que se dispone de un certificado, ya sea mediante pago a una autoridad certificadora o mediante su creación para entornos de prueba, comienza el auténtico proceso de firmado, que se basa en sencillos pasos utilizando las herramientas del *SDK* para *Java* de *Microsoft*. Los pasos para generar un *applet* firmado desde este estado son:

CREAR EL FICHERO CAB

Para realizar el cifrado es necesario que todas las clases relacionadas estén agrupadas en un fichero *cab*, que no es más que un archivo que agrupa a otros archivos y los comprime.

Siempre se recomienda que las clases que se van a empaquetar estén agrupadas en un paquete *Java* determinado, por cuestiones de reutilización en otros procesos, como utilización de *DUBUILD*, etc. En cualquier caso para realizar el proceso de firmado no es estrictamente necesario.

Conviene recordar que para poder compilar el código fuente es necesario disponer de las clases *Microsoft* del paquete *com.ms.security*. Si no se desea instalar las clases propias del *SDK* de *Java* es posible obtenerlas del archivo **Zt7tjlnl.zip** que se crea con la instalación de *Explorer*. Se encuentra en el directorio *c:\winnt\java\packages*.

Además, estas son las clases que el navegador utiliza cuando ejecuta los *applets*, así que tienes toda la garantía de que si el proceso funciona será correcto en el resto de navegadores.

Para crear el fichero *cab* no tenemos más que invocar la utili-

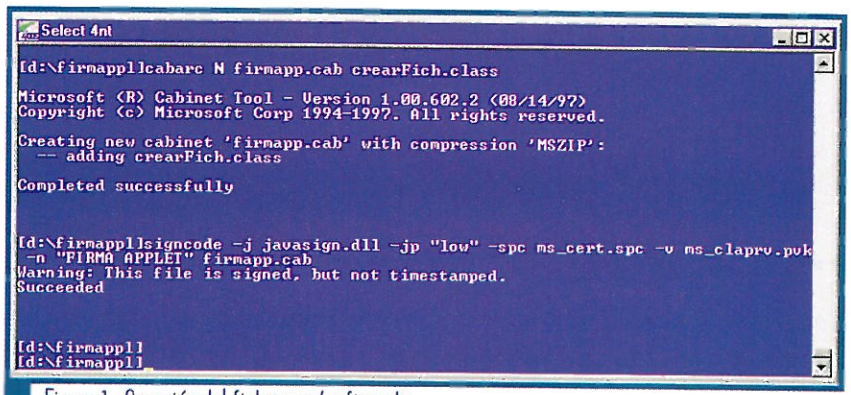


Figura 1.- Creación del fichero *cab* y firmado.

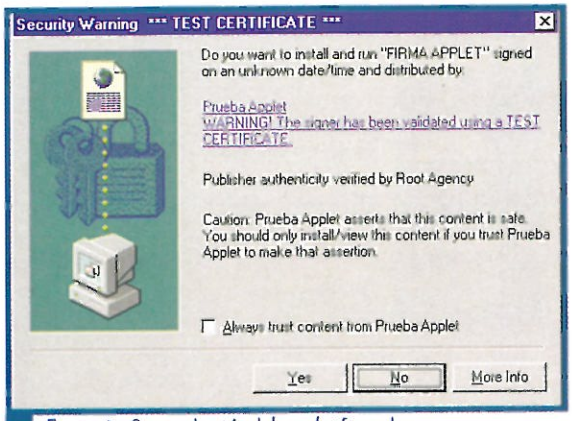


Figura 2.- Comprobación del *applet* firmado.

dad *cabarc* que se detalló en el artículo anterior de la siguiente forma:

```
cabarc -p -r N fichero.cab
    classes
```

El parámetro *N* indica creación de un nuevo fichero *cab* y los parámetros *"-p -r"* se refieren a que se deben incluir los directorios adecuados y por tanto, estilo paquete *Java*. Para nuestro ejemplo, podríamos ejecutar lo siguiente:

```
cabarc N firmapp.cab
    crearFich.class
```

FIRMADO DEL CÓDIGO CONTENIDO EN EL FICHERO CAB

En este paso se realiza el firmado, determinando qué niveles de privilegio se le van a asignar al *applet*.

En *Explorer* se definen tres niveles de seguridad que se denominan *low, medium y high*. Un nivel *high* determina que el *applet* está restringido a la *sand-box*, como si fuera un *applet* sin firmar. Un nivel *medium*, permite el acceso a determinados recursos, como acceso a ficheros temporales, entrada/salida, redirecciones, etc. Un nivel *low* permite saltarse las restricciones de la *sand-box* y acceder a la totalidad de los recursos disponibles.

Llegados a este punto, lo único que hay que tener disponible son los ficheros pertenecientes al certificado, **ms_claprv.pvk** y **ms_cert.spc**. Para realizar el firmado del *cab* ejecutamos el siguiente comando:

```
signcode -j java-
sign.dll -jp nivel
-spc fich.spc
-v fich.pvk -n "DES-
    CRIPCION APPLET" fichero.cab
```

donde *nivel* especifica el nivel de seguridad, *low, medium* o *high*, el prefijo *"-spc"* selecciona el fichero *.spc*, el "–v" el *.pvk* y "-n" es un mensaje descriptivo del *applet*.

Siguiendo nuestro ejemplo tendríamos:

```
signcode -j javasign.dll -jp
    "low" -spc ms_cert.spc
-v ms_claprv.pvk -n "FIRMA
    APPLET" firmapp.cab
```

Tras la ejecución de ese comando se dispone ya del *applet* firmado, aunque el proceso completo aún no ha finalizado. En la Figura 1 se puede observar la ejecución real de estos pasos.

REALIZAR EL TIME STAMPING

El *time stamping* es un proceso que está íntimamente ligado al contexto del *no-repudio* de la firma digital, es decir, a cómo controlar que, una vez que un certificado es válido, lo siga siendo por un periodo de tiempo determinado y por supuesto, que a partir de determinado período deje de tener validez.

En realidad, el proceso en sí es más importante por otra circunstancia. ¿Quién me garantiza a mí que el certificado era válido antes de iniciar el proceso del firmado del *applet*? ¿y si el certificado no era válido en ese momento?. Para realizar este tipo de validaciones, el certificado utilizado para firmar un *applet* va acompañado de una información adicional, el *time stamping*.

Esta información adicional se obtiene mediante la comunicación con una *TSA* (*Time Stamping Authority*) y es el resultado de aplicar una función de *hash*, con lo que en realidad se dispone de más información adicional, como es el nombre de la entidad *TSA* y la fecha de caducidad de la clave privada utilizada para crear el certificado y por tanto, el período de validez de este último.

Pasando a la parte práctica, para realizar este proceso necesitamos conexión a *Internet* y una entidad *TSA*. *Verisign* ofrece este servicio gratuitamente, por lo que para utilizarlo en nuestro firmado se tiene que ejecutar el siguiente comando:

```
signcode -x -t
    http://timestamp.verisign.com
    /scripts/timstamp.dll
      -tr 5 fichcabfirmado.cab
```

donde el parámetro *–x* indica que se debe únicamente realizar el *timestamping*, no el firmado. El parámetro *–t* indica el servidor a utilizar, *-tr* el número de reintentos y se finaliza con nuestro *applet* firmado. En nuestro ejemplo, simplemente tenemos que ejecutar:

```
signcode -x -t
    http://timestamp.verisign.com
    /scripts/timstamp.dll
      -tr 5 firmapp.cab
```

Si se dispone de conexión a *Internet*, en una ventana del *DOS* se ejecuta el comando, se espera unos segundos y si todo ha ido bien, el cursor retornará sin mostrar ningún mensaje a la línea de comandos.

COMPROBACIÓN DEL FIRMADO OBTENIDO

Llegados a este punto, nuestro *applet* ya está disponible para ser utilizado, por lo que únicamente queda probarlo y verificar si todo se ha realizado correctamente. Para realizar este proceso se puede utilizar el comando *chktrust*, que mostrará una ventana como la que indica la Figura 2.

A partir de este momento, si todo es válido, sólo queda probar la ejecución real de todo el proceso.

PROBAR EL APPLET FIRMADO

Una de las grandes diferencias entre los navegadores es la forma en que reconocerán los *applets* firmados. En el caso de *Explorer*, para que reconozca el *applet* debe ir incluido dentro de una entrada *param* de una sentencia *applet* en *HTML*. De tal forma que para poder probar el *applet* firmado debemos crear una página del siguiente estilo:

```
<applet code="crearFich.class"
     width=400 height=400>
     <param name="CABBASE"
     value="firmapp.cab">
</applet>
```

donde **param name="CABBASE"** indica al navegador que tiene que buscar la clase principal de ejecución en el fichero indicado con la entrada *value* correspondiente, en este caso **value="firmapp.cab"** que contiene nuestro *applet* firmado.

Una vez que comienza la ejecución del *applet* aparecerá una ventana indicando que se va a ejecutar un *applet* que solicita permisos totales (ver Figura 3). Si confirmamos su ejecución, encontraremos que este *applet* ha añadido un acceso directo en nuestro escritorio a un archivo con unos cuantos caracteres de texto. Todo esto ocurre con total transparencia, por lo que el usuario debe tomar conciencia de que asentir la ejecución con determinados permisos a un *applet* es como ofrecer nuestro ordenador a un programa ajeno.

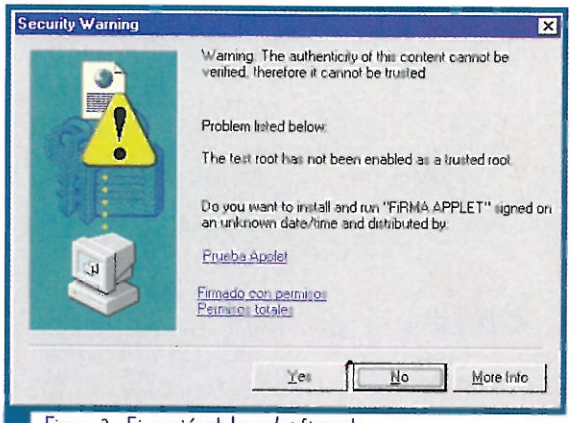


Figura 3.- Ejecución del *applet* firmado.

Gracias a esta facilidad se pueden aplicar nuevas funcionalidades, como es la instalación de paquetes de clases de forma local, ideal cuando el *applet* es voluminoso y no se quiere descargar un amplio conjunto de clases, lo que podría incurrir en un tiempo de carga elevado y en el aburrimiento del usuario.

FIRMANDO DE APPLETS PARA NAVEGADORES NETSCAPE

El proceso de firmado de *applets* para *Netscape* siempre ha estado rodeado de un cierto conjunto de comentarios negativos respecto a su posibilidad de uso y facilidad de implementación.

La principal razón por la que la mayoría de los desarrolladores piensan que firmar los *applets* para *Netscape* es más complicado que para *Explorer* es debido a que las herramientas disponibles para ello estaban escasamente documentadas, el proceso parecía algo más complicado y la diversidad de versiones y entornos disponibles hacían más complicada la comprobación de una correcta ejecución.

PROCESO DE FIRMADO

El proceso de firmado sigue las mismas pautas que para *Explorer*, es decir, modificar el código del *applet* para definir las zonas que realizan acciones prohibidas, obtener un certificado, ya sea mediante pago o por la creación personal con herramientas gratuitas disponibles para ello, importar el certificado creado y comprobar su funcionamiento.

DESARROLLO DE ACCIONES PRIVILEGIADAS

Si desarrollamos un *applet* que realiza acciones prohibidas, el navegador, como es de esperar, nos indicará mediante el lanzamiento de una excepción del tipo *nestcape .security.AppletSecurityException* que se han intentado realizar acciones sin permisos adecuados.

Para que los *applets* puedan realizar acciones prohibidas en *Netscape* hay que utilizar las *Netscape's Capabilities API* como se mencionó en el anterior artículo. Gracias a esta *API* se pueden especificar qué tipo de acciones se pueden realizar en función del nivel de privilegio que se necesite.

Existen muchos y bien diferenciados niveles, que a su vez se agrupan en tres niveles de riesgo:

LISTADO 1. Ejemplos en Netscape

```
import java.applet.*;
import java.awt.*;
import java.io.*;
import netscape.security.PrivilegeManager;

public class crearFich extends Applet
{
   public void paint ( Graphics g )
      {
        try
  {
     String tipo= System.getProperty("java.vendor");

     if ( tipo.indexOf ( "Netscape") != -1 )
        {
                PrivilegeManager.enablePrivilege
("UniversalFileAccess");
        }

     g.drawString ( "Capturando ..",10,10);
     String userdir= System.getProperty("user.dir");
     String slash= System.getProperty("file.separator");

     g.drawString ( "Leidas propiedades", 10,50);

     String fich = userdir + slash + "tmpfoo";

     DataOutputStream dos;
     dos = new DataOutputStream (
     new FileOutputStream ( fich ));

     dos.writeChars ("Escribiendo cars.");
     dos.close();
     g.drawString ("He escrito " + fich, 10,90);
      }
  catch ( Exception e )
          {
        g.drawString ( "crearFich Exception " + e, 10, 90);
    }
      }
}
```

high, medium y low. Sin embargo, el sistema aplicado en este caso es contrario semánticamente al de *Explorer*, puesto que *high* significa que pueden realizar las tareas de mayor riesgo para el cliente, luego mayor libertad para el *applet*. Se pueden abrir *sockets*, crear ficheros, etc. El nivel *medium* reduce algunas de las anteriores posibilidades, pero aún con ello se pueden crear archivos, enviar correos, etc. El nivel *low* es el de mayor seguridad para el usuario y de menor libertad para el

applet, puesto que se limita prácticamente a las opciones de la *sand-box*. Para una completa descripción de los niveles de privilegio y acceso se puede consultar la *Web* de *Netscape*, *http://developer.netscape.com*.

Para nuestro ejemplo, se requiere un tipo de privilegio denominado "*UniversalFileAccess*", de nivel *high*, lo que nos permitirá leer propiedades del sistema local y crear archivos. En el Listado 1 se puede observar la adaptación de nuestro ejemplo para ser utilizado con *Netscape*.

Para poder compilar el código es necesario añadir el fichero **java40.jar** en el *classpath* de nuestro sistema. Este fichero se sitúa en:

```
C:\ProgramFiles\Netscape\Communicator\Program\java\classes
```

en una instalación normal, sin modificaciones.

Una vez cumplido este requisito, revisamos el código propiamente dicho. La siguiente línea de código es la más importante:

```
PrivilegeManager.enablePrivilege("UniversalFileAccess");
```

Con ella se indica que se va a realizar determinado conjunto de acciones, indicando el nivel de privilegio, y a partir de que el usuario conceda su validez al certificado, el *applet* podrá realizar las tareas prohibidas.

Ahora bien, también hay que realizar otros cambios. El paquete de las *capabilities* que se debe utilizar es *netscape.security*. En *Netscape* se pueden realizar esas tareas por ejemplo en el método *paint*() tal y como se observa en el ejemplo, pero nos encontraremos con otro aspecto a tener en cuenta. Al crear el archivo y conocer el directorio por defecto del usuario, debemos conocer qué tipo de sistema de ficheros se utiliza para realizar la concatenación del nombre del fichero con la del directorio.

Como *Netscape* puede ser utilizado en entornos *Unix* y *Windows*, la concatenación debe tener en cuenta distintos símbolos, "\" para *Windows* y "/" para *Unix*. Hay que añadir código para tener esto en cuenta. Es simplemente solicitar una propiedad, "*file.separator*", que no está prohibida por defecto.

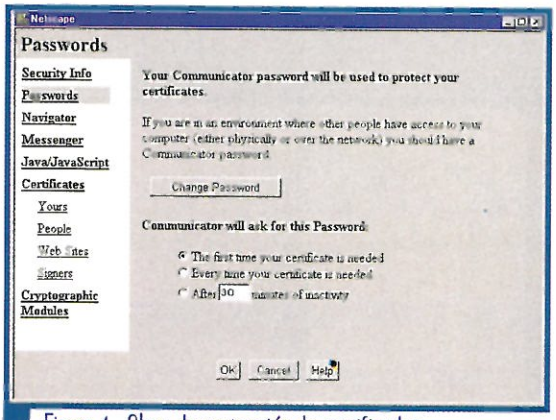


Figura 4.- Clave de protección de certificados.

HERRAMIENTA A UTILIZAR

Para todos los procesos relativos a la firma bajo *Netscape* hay que utilizar una herramienta denominada *signtool*. En la *Web* de *Netscape* se puede descargar en las diferentes plataformas existentes, por ejemplo, encontrará el archivo **signtool11WIN95.zip** para *Windows 95/NT*. Dentro únicamente se encuentra la utilidad *signtool.exe* que puede realizar múltiples tareas, como crear certificados, firmarlos, listarlos, etc.

CREANDO UN CERTIFICADO

Para comenzar la creación de un certificado hay que realizar unas tareas mínimas. Se tiene que copiar en un directorio de pruebas los archivos **cert7.db** y **key3.db** que se encuentran en el directorio:

```
C:\ProgramFiles\Netscape\Users\<usuario>
```

ya que serán utilizados para crear certificados.

Además, hay que poner una clave de protección a los certificados del navegador. Para realizar esta tarea, se selecciona en el menú del navegador la opción **Communicator->Tools->SecurityInfo->Passwords**, se pulsa el botón de activar *password*, se introduce una clave secreta propia y se finaliza.

En la Figura 4 se muestra la ventana donde se realiza la acción. Una vez introducida la clave, se presentará la misma ventana con la única diferencia de que el botón indicará la posibilidad de cambiar la clave.

Una vez realizadas estas dos sencillas operaciones, ya se está preparado para crear un certificado propio. Para ello se utiliza la herramienta *signtool*. Se abre una ventana de *DOS* y se ejecuta el siguiente comando:

```
signtool –G"mensaje descriptivo"
    –d"directorio clases"
```

donde la opción *–G* especifica el nombre o identificación del certificado y *–d* el directorio que contiene los ficheros *.db*.

```
d:\firmappl\netscape\signtool.exe
    –G"clapri" –d"."
```

Donde **clapri** es la denominación para el certificado y **"."** el directorio actual que consta de los ficheros *.db* .

A continuación tendremos que introducir una serie de datos relacionados con la creación del certificado, tales como un nombre para el mismo,

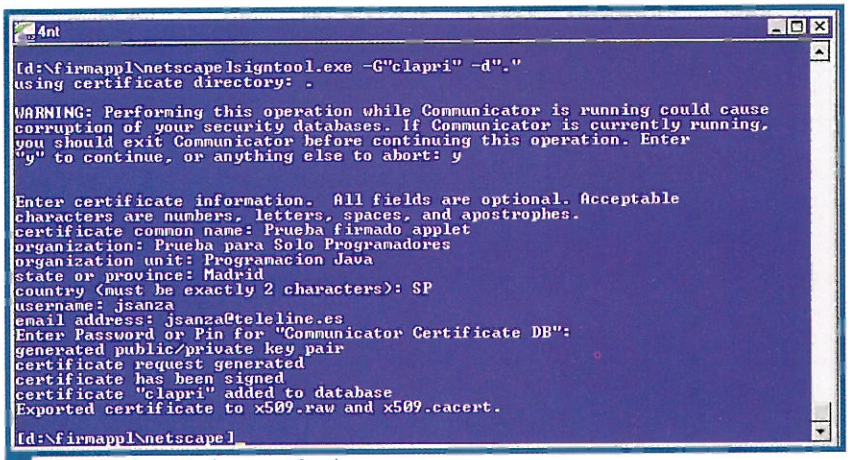


Figura 5.- Creación de un certificado.

la organización a la que representa, usuario, cuenta de correo, etc. En la Figura 5 se muestra el proceso de creación de un certificado.

Al finalizar este proceso, y si todo ha ido bien, se creará un fichero denominado **x509.cacert** en el directorio donde se ejecutó *signtool*. Este fichero es el clásico certificado tipo *x509*, que es el estándar más utilizando entre los navegadores de *Internet*.

IMPORTACIÓN DEL CERTIFICADO

Una vez que el certificado ha sido creado se necesita importar al navegador que lo va a utilizar, de tal forma que este último reconozca los *applets* que estén firmados con él. El método más sencillo consiste en crear una página *HTML* que contenga un enlace al certificado, algo del estilo:

```
<a href="x509.cacert"> Pulsar
    para importar </a>
```

Para que la extensión *.cacert* sea reconocida por *Netscape* hay que añadir un nuevo tipo *MIME* al mismo. Para ello iremos a la opción de menú **Editar->Preferencias->Navigator->Aplicaciones**. Buscaremos la entrada *application/x-x509-ca-cert* por si existiera, y en caso contrario tendríamos que crearla. Para ello simplemente se deberá pulsar en **Crear nuevo tipo** e introducir los siguientes parámetros:

```
Descripción : x509 CA certificate
Extension   : cacert
MIME tipo   : application/
   x-x509-ca-cert
Aplicación  :
   c:\winnt\system32\rundll32.exe
   c:\winnt\system32\inetcpl.cpl,
   SiteCert_
   RunFromCmdLine %1
```

A continuación, y habiendo puesto la página *HTML* en un servidor *Web* de prueba, nos conectaremos a la misma. Si al pulsar sobre el hipervínculo únicamente se muestra el certificado, es que el servidor *Web* no entiende el tipo *MIME* que acabamos de añadir al navegador.

En función del servidor *Web* de que se trate, este proceso se realizará de una forma u otra. En la mayoría de los servidores existe un fichero denominado *mime.types* en el cual basta con añadir una línea *cacert=application/x-x509-ca-cert* y rearrancar el servidor para que todo el proceso funcione correctamente.

Para saber que todo funciona bien, nuestro navegador mostrará una ventana, como la de la Figura 6, en la que se indica que se inicia el proceso de instalación de un nuevo certificado.

Siguiendo las instrucciones del asistente nuestro certificado es importado. Para comprobarlo a ciencia cierta, podemos utilizar *signtool* e indicar que nos muestre la lista de todos los certificados instalados. Para ello, se dispone de la opción *"–l"*. La lista es reducida porque no están incluidos los de las autoridades de certificación. Para ver todos los certificados hay que utilizar *"-L"*, y los obtenidos mediante *"-l"* aparecerán con un "*" a la izquierda.

CREANDO UN JAR FIRMADO

Una vez que el certificado se ha importado correctamente hay que empaquetar las clases de nuestro *applet* y firmarlo. Para ello, las clases de nuestro *applet* crearán un fichero *JAR* (*Java ARchive*) y éste será firmado. Utilizamos la herramienta *signtool* que requiere de una serie de opciones que se muestran a continuación:

- *dtexto*, directorio que tiene los certificados.
- *ktexto*, identificación de nuestro certificado.
- *ptexto*, clave de protección de los certificados.
- *Ztexto*, nombre del fichero *JAR*.
- *cnum*, nivel de compresión.
- *dir*, directorio que contiene las clases.

De tal forma que el comando sería el siguiente:

```
signtool -d"." -k"clapri"
-p"hola" -Zfirmapp.jar -c0
    firmado
```

La ejecución resultante sería la siguiente:

```
using certificate directory: .
Generating firmado/META-INF/
```

Figura 6.- Importación del certificado.

```
    manifest.mf file..
--> crearFich.class
adding firmado/crearFich.class to
    firmapp.jar...(deflated 0%)
Generating zigbert.sf file..
adding firmado/META-INF
    /manifest.mf to
    firmapp.jar...(deflated 0%)
adding firmado/META-INF/zigbert.sf
    to firmapp.jar...(deflated
    0%)
adding firmado/META-INF
    /zigbert.rsa to
    firmapp.jar...(deflated 0%)
tree "firmado" signed success-
    fully
```

Si durante la compilación del programa de ejemplo encuentras ciertos errores con las clases, extrae el paquete *netscape.security* a su directorio de compilación y elimina *java40.jar* del *classpath*. Existen determinadas versiones de navegadores que tienen ciertos problemas de nombrado en las clases. Con este sencillo método se pueden evitar al compilar.

En este momento ya disponemos de nuestro *applet* firmado para utilizarlo en cualquier navegador de *Netscape* en sus diferentes plataformas.

VERIFICANDO EL APPLET FIRMADO

Antes de verificar el *applet* es necesario comprobar que no se tenga ningún navegador ejecutándose. Para comprobar un *applet* firmado se usa *signtool* con el siguiente formato:

```
signtool -d"dir_fich_db" -w
    fich.jar
```

es decir, "*-d*" indica el directorio que contiene los ficheros *.db* y "*–w*" indica el fichero *JAR* firmado.

Siguiendo nuestro ejemplo, se tendría:

```
signtool -d"." -w firmapp.jar
```

El resultado que se obtendría por pantalla sería el siguiente:

```
using certificate directory: .

Signer information:

        nickname: clapri

subject name: CN=Prueba
firmado applet, O=Prueba
para Solo Programadores,
OU=Programacion Java,
ST=Madrid, C=SP, UID=jsan-
za, E=jsanza@teleline.es
issuer name: CN=Prueba
firmado applet, O=Prueba
para Solo Programadores,
OU=Programacion Java,
ST=Madrid, C=SP, UID=jsan-
za, E=jsanza@teleline.es
```

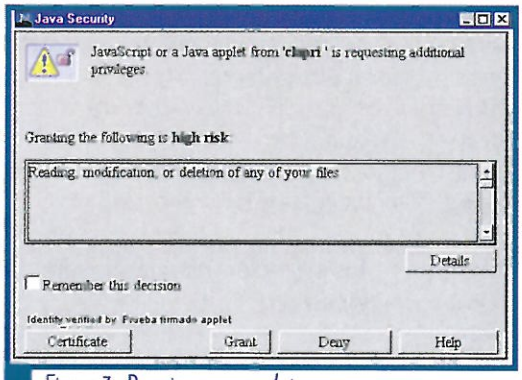


Figura 7.- Permiso para *applet*.

PROBAR EL APPLET FIRMADO

Para comprobar el *applet* no hay mas que crear una página *HTML*, situarla en el directorio correspondiente de nuestro servidor *Web* y conectarnos a él solicitando la misma. La página *HTML* puede ser la siguiente:

```
<applet code="crearFich.class"
     width=400 height=400
     archive="firmapp.jar">
</applet>
```

En la entrada *archive* se indica cuál es el fichero *jar* que contiene la clase *crearFich.class*. Una vez se carga la página, el navegador muestra la siguiente ventana (Figura 7), pidiendo permiso para ejecutar el *applet* firmado.

Si se le permite, el *applet* obtendrá los permisos necesarios para ejecutarse, por lo que conseguiremos escribir un archivo en el disco local. En la Figura 8 puede apreciarse la ejecución del ejemplo.

Como puede observarse, el proceso con *Netscape* es algo más tedioso y con mayor posibilidad de cometer errores. De todos modos, la gran ventaja es la sencillez en el código, y lo más importante, la ejecución multiplataforma, puesto que existen navegadores *Netscape* para múltiples entornos.

EJECUCIÓN EN MÚLTIPLES NAVEGADORES

Hasta ahora se ha mostrado cómo poder ejecutar nuestro *applet* firmado en un determinado tipo de navegador, pero esta circunstancia

no tiene por qué ser válida en todos las aplicaciones, es decir, ¿existe alguna forma para poder ejecutar el *applet* firmado con independencia del navegador, tanto en *Explorer* como en *Netscape*?.

La contestación a esta pregunta es sencilla, y en la mayoría de los casos es afirmativa. Se puede solucionar esta situación de la manera más simple, creando los certificados para cada navegador como se ha mostrado hasta ahora, y añadiendo dos cambios:

- El código de nuestro *applet* debe incluir referencias para diferenciar el navegador sobre el que se está ejecutando.
- La página *HTML* que contiene la carga del *applet* debe construirse para poder diferenciar la carga del *applet* en función de un *.jar* o un *.cab* según el navegador.

Solucionar la primera parte es relativamente sencillo aunque provoca que el código quede un poco engorroso. Para ello, simplemente hay que hacer los *imports* para los paquetes de ambos navegadores, es decir:

```
import
    netscape.security.PrivilegeMa
    nager;
import com.ms.security.*;
```

A continuación, para diferenciar el navegador, utilizamos código como el siguiente:

```
String tipo=
    System.getProperty("java.
    vendor");
if ( tipo.indexOf ( "Netscape")
    != -1 ) {

    PrivilegeManager.enablePrivil
    ege ("UniversalFileAccess");
 }
else
 {
  if ( tipo.indexOf (
    "Microsoft") != -1 ) {
   PolicyEngine.assertPermission
    ( PermissionID.SYSTEM);
 }
}
```

Como se aprecia, se utiliza un simple modo para detectar el tipo de navegador y en función de uno u otro se aplica la correspondiente forma de acceso a acciones prohibidas.

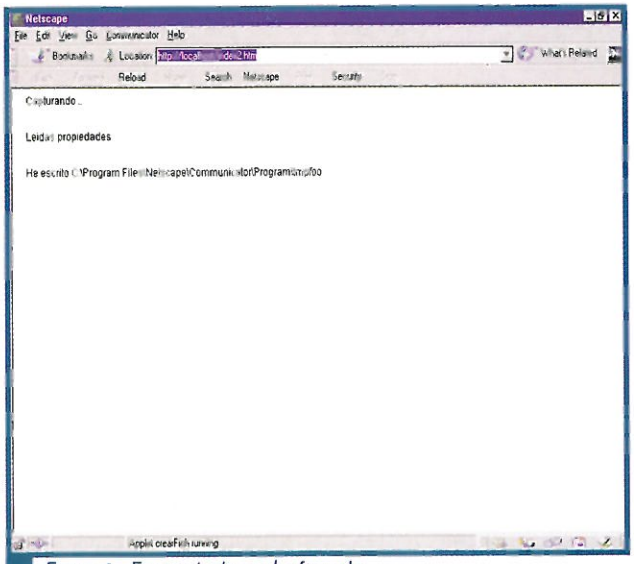


Figura 8.- Ejecución de *applet* firmado.

Más sencilla aún es la creación de la página *HTML* que permite la carga del *applet* firmado en un *.cab* o en un *.jar*. El siguiente código permite la carga en función del navegador. Cuando es *Netscape*, éste entenderá el *tag archive* e invocará a *crearFich*. En el caso de *Explorer*, éste mirará en el *tag name="cabbase"* y obtendrá el contenido del correspondiente *value*, que es el *cab* firmado.

```
<applet code="crearFich.class"
     width=400 height=400
     archive="firmapp.jar">
     <param name="CABBASE"
     value="firmapp.cab">
</applet>
```

Se puede apreciar que tampoco es un proceso muy complicado, pero por regla general, se suele determinar el navegador por defecto a utilizar en un tipo de aplicación para eliminar posibles cambios de versiones, etc.

CONCLUSIÓN

A lo largo de este artículo se ha presentado cómo realizar el firmado de *applets* en los navegadores *Netscape* y *Explorer*, que son los mayormente utilizados.

El proceso con *Explorer* es relativamente sencillo, y si se siguen los pasos adecuadamente, el conjunto de errores que se pueden producir se puede detectar mediante una serie de sencillas revisiones. Además, con el firmado se pueden utilizar posibilidades avanzadas como la instalación local de paquetes.

Al utilizar *Netscape* encontramos que el proceso parece algo menos automático, puesto que exige de un conjunto mayor de pasos y algo más engorrosos, pero de todos modos no deja de ser casi automático en cuanto se realiza unas cuantas veces. La ventaja de firmar *applets* con *Netscape* es la multiplataforma, puesto que se podrá ejecutar en diferentes entornos.

Programación OpenGL en Delphi (I)

Juan Luis Ceada Ramos.
Diplomado en Informática de Sistemas.
Emilio Nestal Díaz.
Ingeniero Técnico en Informática.

Iniciamos en este número una nueva serie de artículos dedicados a algo que se está poniendo muy de moda últimamente, la programación gráfica, en concreto al estándar *OpenGL (Open Graphics Library)* desarrollado por *Silicon Graphics.*

INTRODUCCIÓN

OpengGL no es más que una interfaz que permite representar imágenes *3D* muy fácilmente, aprovechando, en caso de que se encuentren disponibles, las capacidades de aceleración gráfica que proporcionan las tarjetas de vídeo.

Esta interfaz ha sido desarrollada por *Silicon Graphics* (en adelante *SGI*), verdaderos expertos en la visualización y manipulación de gráficos en *3D*. La gran experiencia de *SGI* en este campo ha beneficiado a la interfaz *OpenGl*. Los algoritmos que se emplean están altamente optimizados y han sido diseñados para aprovechar las características de las últimas tarjetas de aceleración *3D*.

La API OpenGL permite aprovechar el hardware de aceleración 3D

¿Quiere esto decir que si nuestra tarjeta no proporciona soporte *OpenGL*, no podremos seguir la serie de artículos?. Nada más lejos de la realidad. Para soportar la aceleración por *hardware*, cada fabricante de aceleradoras debe proporcionar los controladores necesarios.

Pero *Microsoft* proporciona con *Windows 9x/NT* una implementación genérica del estándar. Al emular el *hardware* mediante *software*, podremos ejecutar los ejemplos que acompañan a los artículos sin ningún problema en casi cualquier máquina. Eso sí, no esperéis fluidez, ni una visualización perfecta en escenas formadas por muchos objetos.

OpenGL es una norma abierta (como bien indica el "*Open*" de su

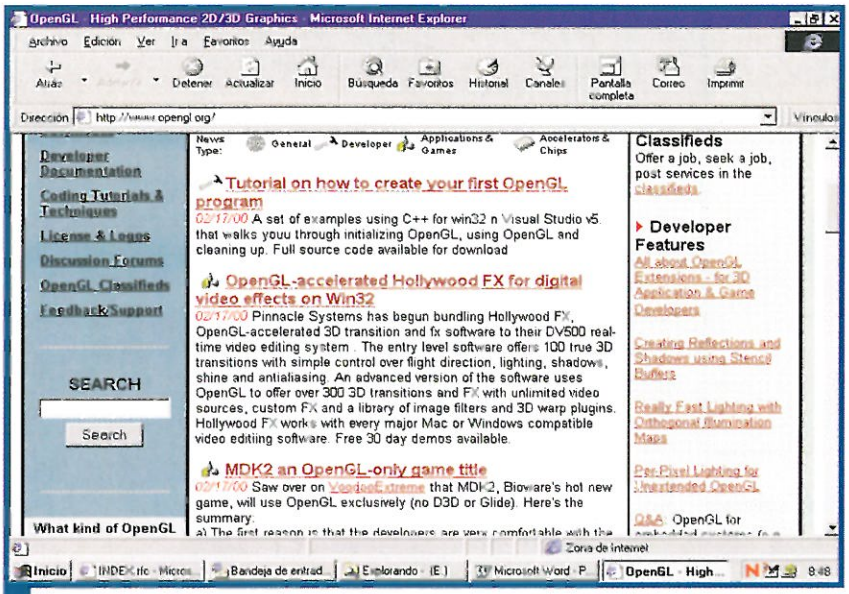


Figura 1.- www.opengl.org

nombre). Por tanto, existen implementaciones para múltiples plataformas, y al contrario que otras alternativas, como *DirectX*, no está controlada por un único fabricante. Todas las mejoras que se realicen son controladas por la Plataforma para las Revisiones de la Arquitectura *OpenGl* (*Open GL Arquitecture Review Board, ARB*), cuyos miembros fundadores son *SGI, DEC, IBM, Microsoft* e *Intel*.

Microsoft proporciona una implementación genérica del estándar

En estos momentos la revisión más moderna disponible es la 1.2. Esto da una idea de lo bien que ha sido diseñado este estándar, ya que desde su fecha de lanzamiento (en Julio de 1992) hasta la actualidad sólo han sido necesarias dos revisiones.

OpenGl soporta, entre otras, el dibujado de primitivas gráficas como puntos, líneas o polígonos en *3D*, iluminación, sombreado, mapeado con texturas, animación, niebla, etc. Además tiene fama de ser muy fácil de programar, como veremos a lo largo de la serie. Basta con ver el excelente trabajo de *John Carmack* en su última creación, *Quake III Arena*. Hace tiempo, intentó portar su saga *Quake* a *DirectX*, pero debido a su excesiva complejidad, decidió usar *OpenGl*, y en poco tiempo tuvo lista la versión *OpenGl* de *Quake* I, *GLQuake*.

Tenemos mucho que agradecer a *John Carmack*. Gracias en parte a las declaraciones que realizó en su tiempo, y al éxito de sus juegos, *OpenGl* ha pasado de ser soportado sólo por tarjetas profesionales a incluirse en cualquier tarjeta aceleradora *3D* que se precie (como las *Riva TNT* o *Voodoo*).

CONCEPTOS SOBRE GRÁFICOS 3D

Antes de comenzar a explicar cómo trabajar con la *API OpenGL*, tendremos que dar unos conocimientos básicos acerca de la generación de gráficos en *3D*. En concreto, veremos los sistemas de coordenadas cartesianas, la generación de imágenes *3D* a partir de *2D*, las proyecciones ortográficas y las perspectivas.

Figura 2.- *Quake III Arena*, basado en el estándar *OpenGL*.

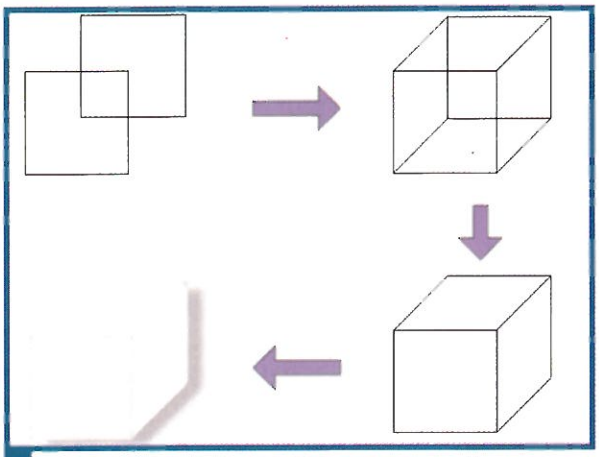


Figura 3.- Un cubo a partir de dos cuadrados.

EL MÁS BÁSICO

Una imagen *3D* representada en la pantalla de un ordenador no es más que una imagen *2D* a la que se le añade la sensación de profundidad, mediante la perspectiva. Dos cuadrados, situados en diferente posición, con sus vértices unidos en un determinado orden, generan un cubo (ver Figura 3).

Ocultando las caras de la parte posterior (que en la vida real no se ven), añadiendo sombras y luces, conseguiríamos un mayor realismo en la representación *3D*.

COORDENADAS CARTESIANAS

Cada uno de los vértices del cubo de la Figura 3 están situados en unas determinadas coordenadas del espacio. Dichas coordenadas se representan mediante tres números, que indican la coordenada X, Y y Z. La coordenada X representa la distancia desde el origen a dicho punto en el eje X (la horizontal). La coordenada Y, es la coordenada vertical y la Z indica la profundidad.

En el sistema cartesiano, el origen de coordenadas está representado por el punto (0,0,0). Un punto con las siguientes coordenadas (5,-6,5) estaría situado a 5 puntos a la derecha del origen, 6 puntos por debajo y a 5 puntos de distancia (ver Figura 4).

En *Windows*, cuando creamos una ventana de 400x300 píxeles, estas serían las coordenadas de sus cuatro vértices (de izquierda a derecha, y de arriba abajo): (0,0), (400,0), (0,300), (400,300). El origen de coordenadas se sitúa en la esquina superior izquierda de la ventana. Sin embargo, al trabajar con *OpenGL*, por defecto el origen de coordenadas se situaría en el centro de la ventana. Ahora las coordenadas de los vértices serían: (-200, 150), (200,150), (-200,-150) y (-200,150).

EL ÁREA DE TRABAJO

Lo que el usuario ve, a veces no es todo lo que hay. Nada nos impide definir una ventana de 400x400 píxeles, y trabajar con un área de trabajo de 1600x1600. Es importante tener clara la diferencia. Si en un área de trabajo pintamos un cubo más grande que nuestra ventana de trabajo, dicho cubo no aparecerá completo ante los ojos del usuario.

Son posibles varias combinaciones. Por ejemplo, una ventana mayor que el área de trabajo, y que lo que se dibuja en el área ocupa toda la ventana. O una ventana mayor que el área, pero lo que haya en el área se ve en la ventana a su tamaño real (aunque sobre parte de la ventana).

PROYECCIONES

Con las proyecciones conseguiremos convertir coordenadas *3D* (el mundo real) a coordenadas *2D* (la pantalla del ordenador), de forma que no se pierda la sensación de que tratamos con imágenes en *3D*.

Es muy fácil realizar aplicaciones basadas en OpenGL

Básicamente, existen dos tipos de proyecciones: ortogonales y en perspectiva. En el primer caso, todas las primitivas que forman un objeto *3D* tienen el mismo tamaño. En la Figura 3, el cubo está dibujado usando este tipo de proyección. Como se puede apreciar, todas las caras tienen el mismo tamaño. Este tipo de perspectiva se suele usar sobre todo en *CAD*. Si mirásemos de frente a dicho cubo, veríamos un cuadrado.

En las proyecciones en perspectiva, lo que está más lejos se ve de menor tamaño. Está es la perspectiva usada en juegos, y es la que

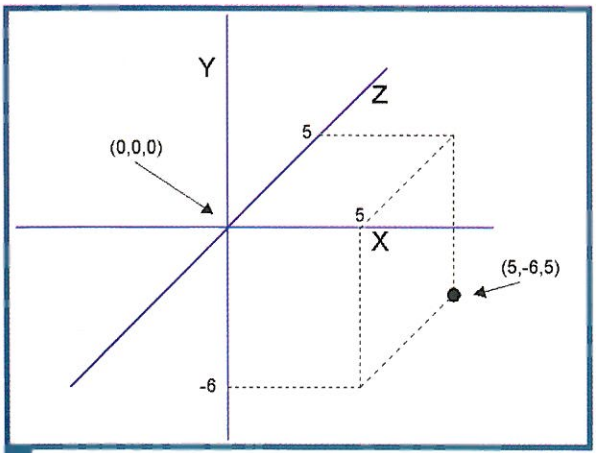


Figura 4.- Coordenadas cartesianas.

existe en la vida real. Si dibujamos un cubo, y luego lo miramos de frente, veremos dos cuadrados, uno dentro de otro.

Pero realmente, ¿cómo consigue *OpenGL* mostrar imágenes *3D* en superficies *2D*?. Supongamos que queremos dibujar un cubo. Pasamos a la *API* seis puntos con coordenadas X, Y, y Z. Con una serie de transformaciones geométricas, basadas en el cálculo matricial, conseguiremos pasar de un sistema de tres coordenadas a dos. Es decir, por cada punto (x, y, z), obtendremos un nuevo punto (x', y'). El valor de z influye directamente en los nuevos x e y obtenidos.

Las matrices con las que se realizan las transformaciones dependen del tipo de proyección seleccionada.

ESTRUCTURA DE OPENGL

Básicamente, la *API OpenGl* está compuesta por dos ficheros. El primero de ellos se corresponde con las especificaciones del *ARB* y su nombre es **opengl32.dll**. En él se almacenan todas las funciones que propone el estándar en su versión 1.2. Todas las funciones incluidas en esta *DLL* tienen el prefijo *gl*.

El segundo es una librería de funciones adicionales diseñadas para hacer la vida más fácil a los programadores. El nombre del fichero es **glu32.dll**. Por ejemplo, gracias a estas funciones podemos dibujar esferas con un única función (que internamente lo único que hace es llamar a una o más funciones del fichero *opengl32.dll*).

TABLA 1. Tipos de datos OpenGl.

Tipo de datos	Descripción
GLbyte, GLShort	Entero de 8 bits.
GLint, GLsizei	Entero de 16 bits.
Glfloat, GLclampf	Coma flotante de 32 bits.
GLdouble, GLclampd	Coma flotante de 64 bits.
Glubyte, GLboolean	Entero de 8 bits sin signo.
GLushort	Entero de 16 bits sin signo.
GLuint, GLenum, GLbitfield	Entero de 32 bits sin signo.

Todas las funciones incluidas en esta *DLL* tienen el prefijo *glu*.

TIPOS DE DATOS

OpenGL define además sus propios tipos de datos, de cara a facilitar la migración de aplicaciones entre plataformas. Podéis ver una lista de los tipos de datos proporcionados en la Tabla 1. Nada nos obliga a utilizarlos (por ejemplo, el tipo *GLByte* es equivalente al *byte* de *PASCAL*), pero es conveniente, si queremos mantener un código medianamente portable.

DENOMINACIÓN DE FUNCIONES

El nombre de todas las funciones *OpenGl* sigue una estructura determinada: prefijo + nombre + número de parámetros + tipo de los parámetros.

Por ejemplo, de un nombre como *glVertex3f*, obtenemos la siguiente información:

- Función definida en *opengl32 .dll* (prefijo **gl**)
- Define un vértice un (**Ver tex**)
- Tiene tres parámetros (**3**)
- Los parámetros son de tipo flotante (**f**)

PRIMEROS PASOS CON OPENGL

Una vez vistos algunos conceptos básicos, empezaremos ahora a desarrollar aplicaciones que hacen uso de la *API OpenGL*.

Para ello vamos a utilizar un *wrapper* de *OpenGL* desarrollado por *Mike Lischke*, y que además de estar incluido en los ejemplos, lo

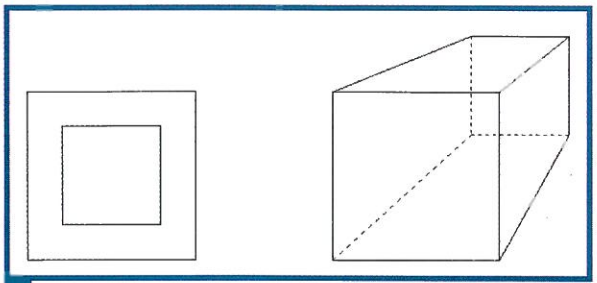
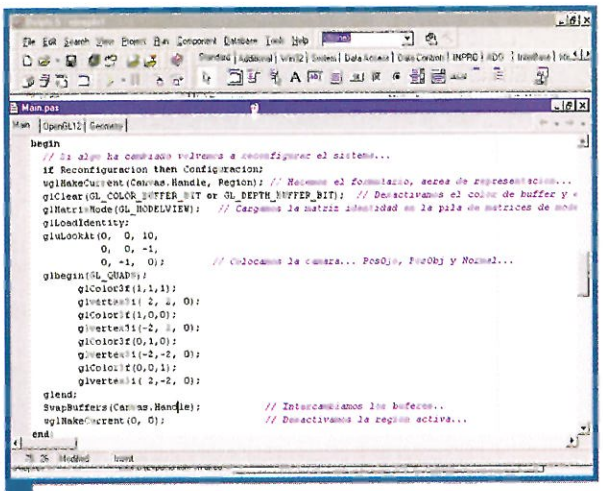


Figura 5.- Proyección en perspectiva.



```
begin
  // Si algo ha cambiado volvemos a reconfigurar el sistema...
  if Reconfiguracion then Configuracion;
  wglMakeCurrent(Canvas.Handle, Region); // Hacemos el formulario, sobre de representación...
  glClear(GL_COLOR_BUFFER_BIT or GL_DEPTH_BUFFER_BIT); // Desactivamos el color de buffer y
  glMatrixMode(GL_MODELVIEW);    // Cargamos la matriz identidad en la pila de matrices de mod
  glLoadIdentity;
  gluLookAt(0,  0, 10,
            0,  0, -1,
            0, +1,  0);          // Colocamos la camara... PosObj, FocObj y Normal...
  glbegin(GL_QUAD);
        glColor3f(1,1,1);
        glvertex3( 2,  2, 0);
        glColor3f(1,0,0);
        glvertex3(-2,  2, 0);
        glColor3f(0,1,0);
        glvertex3(-2, -2, 0);
        glColor3f(0,0,1);
        glvertex3( 2, -2, 0);
  glend;
  SwapBuffers(Canvas.Handle);          // Intercambiamos los buferes...
  wglMakeCurrent(0, 0);                // Desactivamos la region activa...
end;
```

Figura 7.- Código del evento *OnPaint*, encargado de dibujar la escena.

podéis encontrar en *www.lischke online.de.*

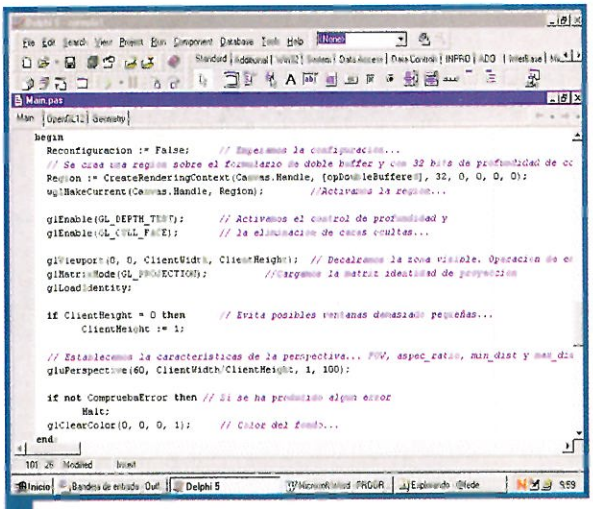
Este *wrapper* no es más que la exportación de todas las constantes, tipos, estructuras y funciones que posee el estándar *OpenGL*, es decir, nos ha ahorrado el trabajo de realizar dichas exportaciones nosotros mismos, y si le echáis un vistazo a dicho *wrapper* lo agradeceréis, dado que el número de exportaciones y definiciones es francamente impresionante.

Por tanto, no vamos a trabajar con ningún tipo de componente, simplemente realizaremos llamadas a la *API OpenGL* de nuestro sistema a través de dicho *wrapper*.

En el ejemplo número 1, que podéis encontrar en el *CD-ROM*, veréis el caso más básico de programa que utiliza *OpenGL* en *Delphi*. En él simplemente aparece un cuadrado en el centro de la pantalla. Dicho ejemplo consta de una serie de detalles de interés, en los que nos vamos a centrar.

Utilizaremos un sistema básico de animación por buffers

Lo primero que debemos realizar es la configuración de nuestro sistema, para ello nos valemos del procedimiento *configuracion*. Este procedimiento es el encargado de, primero establecer la región de la pantalla donde se va a presentar lo que dibujemos, para a continuación establecer las características más básicas de la representación.



```
begin
  Reconfiguracion := False;      // Ingelamos la configuracion...
  // Se crea una region sobre el formulario de doble buffer y con 32 bits de profundidad de co
  Region := CreateRenderingContext(Canvas.Handle, [opDoubleBuffered], 32, 0, 0, 0, 0);
  wglMakeCurrent(Canvas.Handle, Region);      //Activamos la region...

  glEnable(GL_DEPTH_TEST);       // Activamos el control de profundidad y
  glEnable(GL_CULL_FACE);        // la eliminacion de caras ocultas...

  glViewport(0, 0, ClientWidth, ClientHeight);  // Decalcamos la zona visible. Operacion de es
  glMatrixMode(GL_PROJECTION);          //Cargamos la matriz identidad de proyeccion
  glLoadIdentity;

  if ClientHeight = 0 then       // Evita posibles ventanas demasiado pequeñas...
        ClientHeight := 1;

  // Establecemos la caracteristicas de la perspectiva... FOV, aspect_ratio, min_dist y max_dis
  gluPerspective(60, ClientWidth/ClientHeight, 1, 100);

  if not CompruebaError then // Si se ha producido algun error
        Halt;
  glClearColor(0, 0, 0, 1);      // Color del fondo...
end;
```

Figura 6.- Código del procedimiento de configuración.

Por tanto, lo primero que haremos será crear la región de representación mediante la función *OpenGL*, *CreateRenderingContext*. Dicha función toma los parámetros siguientes: una región *canvas* de cualquier objeto, el tipo de configuración de *buffers* de representación, el número de *bits* de color y tres parámetros que, por ahora, no nos importan y que dejaremos a 0. En nuestro caso hemos utilizado: *CreateRenderingContext (Canvas-.Handle, [opDoubleBuffered], 32, 0, 0, 0, 0)*, es decir, la región de representación será el formulario completo, con un doble *buffer* de representación, 32 *bits* color, y el resto a 0 como se indicó anteriormente. Esta función nos devolverá un entero que identificará la región de forma única, es decir, *Region:=CreateRenderingContext(Canvas.Handle,[opDoubleBuffered].*

La configuración del doble *buffer* permite que podamos dibujar en uno de ellos mientras el otro se muestra y evitar así los parpadeos debido a la actualización del *buffer*. Esto nos lleva a que tengamos que copiar el *buffer* de dibujo al *buffer* que se muestra cada vez que cambiemos algo en la escena, es decir, el concepto más básico de animación.

Una vez creada la región de representación, la activaremos mediante, *wglMakeCurrent*, que admite dos parámetros, el *canvas* del objeto y el identificador de la región, es decir, *wglMakeCurrent (Canvas.Handle, Region).*

Creada y activada la región de representación, debemos establecer qué parte de la región vamos a utilizar, mediante la función *glViewport*, que admite cuatro parámetros que definen la sección rectangular de visualización, concretamente: *glViewport(0,0,ClientWidth,ClientHeight),* es decir, toda la ventana principal. No podemos, al menos de forma directa, crear regiones de representación no rectangulares.

A continuación, debemos establecer el tipo de proyección que

mos llamado a esta función debemos comenzar de forma periódica y lo más rápidamente posible a la función *SSND_Poll*, o de lo contrario aparecerán cortes en la reproducción del archivo. Finalmente tenemos una función que permite liberar la memoria que se necesitó para reproducir el *MP3:*

```
int SSND_FreeMPG (void)
```

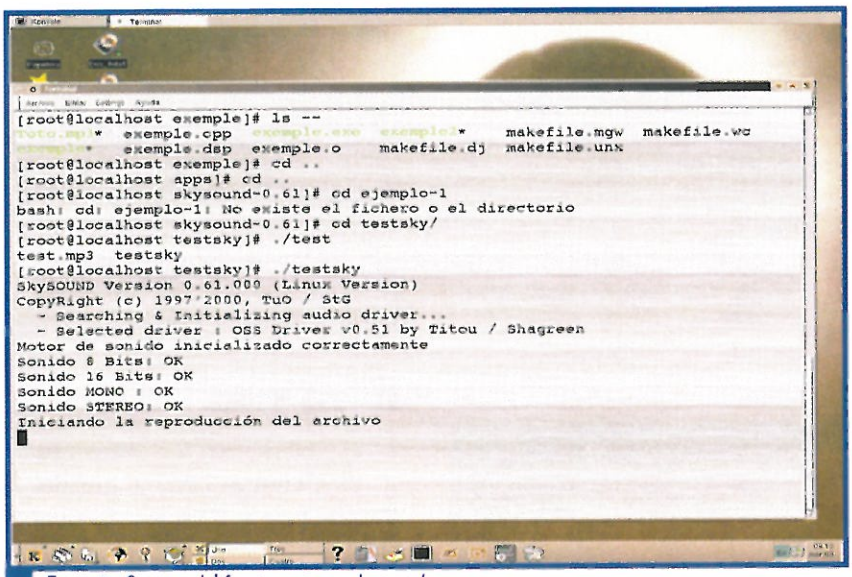


Figura 5.- Captura del funcionamiento de *testsky*.

Esta función libera la memoria consumida por la librería en la reproducción del *MP3*. En el caso de no haber memoria que liberar no se realizaría ninguna acción.

Con estas funciones ya podemos reproducir archivos *MP3* desde nuestros programas, aunque tendremos que adecuarlos para poder incluir algún tipo de llamada periódica a *SSND_Poll*. Un bucle como el que sigue permite reproducir continuamente el *MP3* en muy poco código:

```
do {}
while(SSND_Poll()!=
    SSND_RETURNERROR)
```

Estas funciones sientan las bases de lo que será nuestro pequeño reproductor *MP3* de ejemplo.

BUSCAMOS COLABORADORES

Si además de leer

SÓLO PROGRAMADORES

te gusta la programación, y quieres escribir
en tu revista, no dudes en ponerte en contacto
con nosotros. Envíanos tu propuesta junto a tu curriculum
a la siguiente dirección:

REVISTAS PROFESIONALES
C/San Sotero, 5 - 1.ª planta
28037 Madrid
Ref.: Colaboraciones Sólo Programadores
E-mail: solop@virtualsw.es

COLOCANDO LAS PIEZAS EN SU SITIO

Para ver cómo aplicar estas funciones en nuestros programas utilizaremos un pequeño ejemplo muy limitado pero que incorpora todos los pasos necesarios para realizar la reproducción de archivos *MP3*. Nuestro pequeño programa se llama *testsky*.

La primera función que debemos crear es una que arranque el motor de sonido o salga del programa si no ha podido realizar dicha acción. En nuestro caso hemos creado la siguiente función:

```
void inicializar(void)
{
   if (SSND_Init(SSND_INTER-
    NAL)==SSND_RETURNOK) {
        printf("Motor de sonido
    inicializado
    correctamente\n");
      }
   else {
       printf("El motor de sonido
    no ha sido inicializado\n");
       exit(-1);
   }
};
```

Esta función realiza exactamente la función deseada. El siguiente paso en nuestro ejemplo es detectar las posibilidades de reproducción de nuestro equipo. La función creada simplemente nos informa de las características que están disponibles. Puede verse en el Listado 1.

La reproducción de archivos MP3 es realmente sencilla, en cinco o seis líneas de código podemos hacer que suene un archivo desde el inicio hasta el final

Finalmente hemos creado la función principal en la que primero se prepara nuestro archivo de pruebas para su reproducción y luego se reproduce hasta el final, haciendo un "*polling*" continuo del motor de sonido. Cuando la reproducción termina, el programa libera la memoria consumida y desconecta la librería antes de salir del mismo. El listado completo del código de *testsky* puede encontrarse en el *CD-ROM* de la revista como *test-sky.tar.gz*, junto a la librería utilizada.

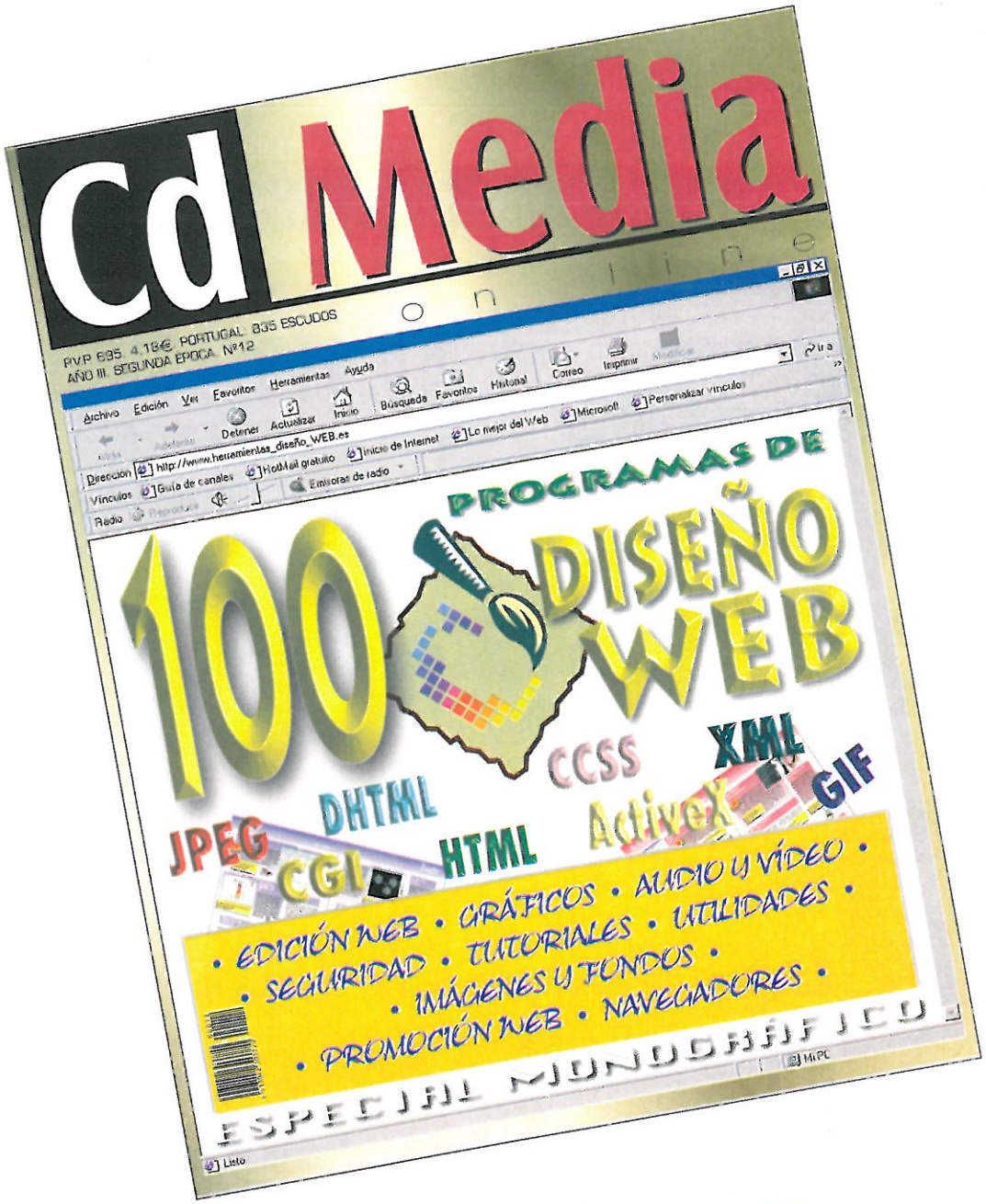
LISTADO 1.

```
void showcap(void)
{
     int cap;

     cap=SSND_GetCaps();

     if (cap&&SSNDCAPS_8BITS) { printf("Sonido 8 Bits: OK\n");
};
     if (cap&&SSNDCAPS_16BITS) {printf("Sonido 16 Bits:
OK\n");};
     if (cap&&SSNDCAPS_MONO) { printf("Sonido MONO : OK\n"); };
     if (cap&&SSNDCAPS_STEREO) {
        printf("Sonido STEREO: OK\n");
     };
};

int main(int argc, char *argv[])
{
         inicializar();
         showcap();
         printf("Iniciando la reproducción del archivo\n");
         SSND_PrepareMPG("test.mp3",0,SSND_MPEGDEFAULT,
               65000,SSND_FROMFILE);
         SSND_PlayMPG();
         do {}
         while (SSND_Poll()!=SSND_RETURNERROR);
         SSND_FreeMPG();
         SSND_Uninit();
         return EXIT_SUCCESS;
}
```

CONCLUSIÓN

En este artículo hemos visto cómo podemos aprovechar la librería *SkySound* para incluir reproducción de archivos *MP3* de forma rápida y sencilla en nuestros programas. *SkySound* no es la librería más flexible, pero sí es cierto que es la más sencilla y cómoda de utilizar. En el siguiente artículo explicaremos el manejo de otra librería que permite realizar la misma operación, pero con un mayor grado de libertad y flexibilidad.

TODOS LOS MESES EN TU QUIOSCO

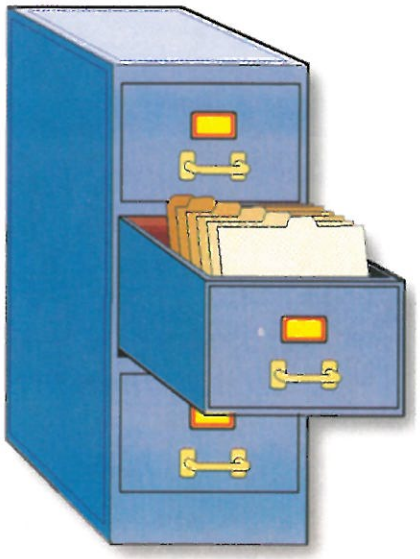


PARA ESTAR AL DÍA EN SOFTWARE SIN GASTARSE UN EURO EN TELÉFONO

LOS MEJORES ESPECIALES
DISEÑO
INTERNET

- MULTIMEDIA
- UTILIDADES
- EN CASTELLANO

- NEGOCIOS
- LO ÚLTIMO
- Y MUCHO MAS...



Aplicaciones de bases de datos con Delphi 5 (III)

Juan Luis Ceada Ramos.
Diplomado en Informática de Sistemas.

Este mes terminaremos de ver algunos de los aspectos teóricos que dejamos pendientes en el número anterior y desarrollaremos una pequeña aplicación donde podremos en práctica los conocimientos adquiridos en artículos anteriores.

ACCESO A CAMPOS

Para poder acceder al contenido de un campo mediante código, tendremos que insertar componentes de acceso a campos. Para ello, bastará con hacer doble *click* en un componente *TTable* o *TQuery*. Aparecerá la pantalla de edición de campos, donde podremos añadir todos los campos de la tabla/consulta, o sólo algunos.

Gracias a estos componentes, podremos controlar las propiedades y eventos asociados a dichos campos. Podremos cambiar las propiedades de visualización, la máscara de edición, etc. También podremos añadir código que compruebe que el valor introducido es correcto (mediante el evento *OnValidate),* por ejemplo.

Los objetos de este tipo tienen un nombre, que se forma mediante la unión **nombre del TDataset + nombre del campo.** Por ejemplo, para acceder al campo **Fecha** de la tabla **Tabla1,** tendremos que usar el identificador **Tabla1Fecha.**

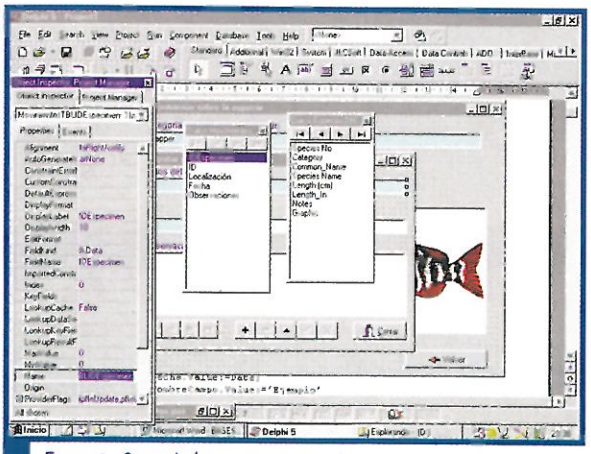


Figura 1.- Creando los componentes de acceso a campos.

Para asignar una serie de valores por defecto, en el evento *On NewRecord,* haríamos lo siguiente:

```
Tabla1Fecha.Value:=Date;
Tabla1NombreCampo.Value:=
    'Ejemplo';
```

Esta forma de acceso a campos es la más segura y eficiente, y presenta la ventaja de que el compilador puede comprobar los tipos de datos. De esta forma evitamos cometer errores de asignación de tipos (como por ejemplo, el que se produce si asignamos a un campo de tipo fecha una cadena).

Los componentes de acceso a campos permiten un acceso seguro y eficiente

También es posible acceder a los campos a través de la tabla a la que pertenecen, por medio de las propiedades *FieldByName* y *Fields*. Un par de ejemplos:

```
Tabla.FieldByName('Fecha').Value
Tabla.Fields[0].Value
```

Son útiles en algunos casos, pero es aconsejable evitar su uso en la medida de lo posible. Si el campo no existe (en el caso del *FieldByName),* o la estructura de la tabla cambia (en el caso de *Fields),* pueden producirse errores inesperados. Además, nada nos impide asignar una cadena a un campo numérico, puesto que el compilador no puede detectar el error en tiempo de compilación.

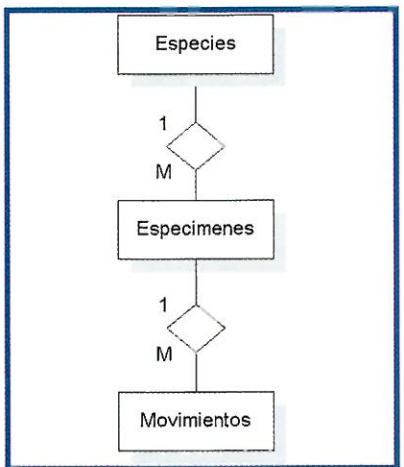


Figura 2.- Diagrama entidad-relación de la aplicación de ejemplo.

RELACIONES MASTER/DETAIL

Las relaciones *master/detail* entre componentes *TDataset* permiten implementar las relaciones de tipo uno a muchos del modelo entidad/relación. Un ejemplo típico de estas relaciones se da en la facturación. Una tabla de facturas tiene asociada una tabla de artículos. En cada factura (un registro en la tabla de facturas), se incluyen varios artículos (varios registros de la tabla de artículos).

En nuestra aplicación *Delphi,* tendríamos dos componentes *TTable* (y sus correspondientes *TDataSource).* Uno enlazaría con la tabla **Facturas** y el otro con la de **Artículos.** La propiedad *MasterSource* de la tabla **Artículos,** enlazaría con el *TDatasource* de la tabla **Facturas.** La propiedad *MasterFields* almacenaría los campos comunes a ambas tablas, de tal forma que así se establecería la relación.

Si ahora utilizamos dos *TDBGrids* y los enlazamos a los *TDataSource,* tendremos nuestra relación uno a muchos implementada. Si nos movemos por la tabla maestra, automáticamente cambian los registros de la tabla hija. Si estamos por ejemplo, en la factura 2, en el segundo *TDbgrid* se muestran todos los artículos que incluía la factura 2. Si cambiamos a la factura 4, se muestran los artículos de dicha factura.

MÓDULOS DE DATOS

Un módulo de datos no es más que un tipo especial de formulario. Se crea mediante la opción del menú **File->New.** Dicho formulario no aparece por ningún lado en tiempo de ejecución. En tiempo de diseño permite agrupar todos los componentes de acceso a datos en un único formulario, no pudiéndose insertar ningún otro tipo de componente.

Un DataModule permite agrupar todos los componentes de acceso a datos

Se puede acceder a los componentes que se incluyen en dicho formulario desde cualquier otro, sin más que incluir el nombre del fichero que representa al *TDataModule* en la cláusula *uses* del resto de los formularios.

Evidentemente, no es obligatorio usar los *TDataModules*. Sigue siendo posible insertar los componentes de acceso a datos dentro de cada formulario. Pero los *TDataModules* permiten centralizar todos los componentes, de forma que podamos compartirlos entre diferentes formularios.

Además, con *Delphi 5* podemos usar los *TDataModules* para documentar nuestros proyectos, gracias a la representación visual de las

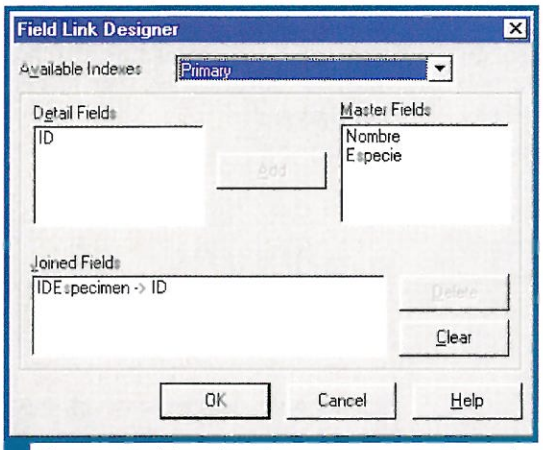


Figura 3.- Estableciendo qué campos nos permitirán crear la relación *master/retail*.

relaciones entre componentes (y por tanto, entre tablas/consultas).

DESARROLLO DE LA APLICACIÓN

Pasaremos ahora a desarrollar una pequeña aplicación de ejemplo. En ella, implementaremos un sistema de seguimiento de animales marinos. Básicamente, trabajaremos con tres tablas. La relación existente entre ellas se muestra en la Figura 2.

La primera de ellas, la tabla de **Especies,** se corresponde con la tabla *biolife.db,* del alias *DBDEMOS.* En ella se encuentran almacenados los datos característicos de varias especies de animales marinos.

La segunda tabla es la de **Especimenes** y tendremos que crearla nosotros mismos. Consta de tres campos, cuyos tipos y descripciones se pueden apreciar en la Tabla 1. Almacena los nombres que los científicos dan a determinados ejemplares, así como la especie a la que pertenecen (por ejemplo Rosa, la Ballena Gris). Es posible que existan varios especímenes de una misma especie.

Podemos usar los DataModules para documentar nuestros proyectos

La tercera tabla es la de movimientos. Podemos ver su estructura en la Tabla 1. En ella se recogen cuales han sido los últimos avistamientos de un determinado espécimen. Por ejemplo, Rosa la Ballena ha sido vista en las costas gallegas, después en las de Portugal, y por último en las de Huelva.

TABLA 1. Estructura de las tablas Especímenes y Movimientos.

Nombre de campo	Tipo	Tamaño	Tabla
ID	Auto		Especimenes.db
Nombre	String	30	Especimenes.db
Especie	String	15	Especimenes.db
IDEspecimen	Long Int		Movimientos.db
ID	Auto		Movimientos.db
Localizacion	String	30	Movimientos.db
Fecha	Date		Movimientos.db
Observacion	Memo	20	Movimientos.db

Todas las tablas se almacenan en el directorio al que apunte el alias **DBDEMOS.** Consulte el fichero **LEEME.TXT** contenido en el *CD-ROM* antes de probar la aplicación de ejemplo.

EL DATAMODULE

Crearemos un *TDataModule,* al que llamaremos **Datos,** y le daremos como nombre de fichero **UDatos.pas.** En él incluiremos tres componentes de tipo *TTable* y sus tres *TDatasources* asociados. Además, añadiremos un *TQuery,* que usaremos para realizar alguna consulta. Todos utilizan el alias *DBDEMOS* (propiedad *AliasName),* y en tiempo de diseño tienen la propiedad *Active* a *True.*

No es mala idea abrir las tablas en tiempo de ejecución

No es mala política abrir las tablas/consultas en tiempo de ejecución. Así nos aseguramos que todo funciona correctamente, aunque olvidemos activar las tablas en tiempo de diseño.

Además, crearemos una relación *master/detail* entre la tabla **Especimenes** y **Movimientos.** Para ello, en el componente **MovimientosTBL,** asignaremos a la propiedad *MasterSource* el valor **EspecimenDS.** Después haremos doble *click* en la propiedad *MasterFields.*

Aparecerá el cuadro de diálogo que se muestra en la Figura 3. En él podemos establecer qué campos se utilizarán para crear la relación *master/detail*. En este caso, enlazaremos el campo *ID Especimen* de la tabla **Movimientos** con el campo *ID* de la tabla **Especimenes** y pulsaremos

OK. Ahora ya tenemos una relación *master/detail* definida.

FORMULARIO PRINCIPAL

En el formulario principal incluiremos los controles que se muestran en la Figura 4. De izquierda a derecha y de arriba a abajo tenemos un *TDBEdit, TDBLookUpCombo Box, TSpeedButton, TDBGrid, TDB Navigator* y varios *TBitBtn*.

El *TDBEdit* enlaza con el campo Nombre de la tabla **Especimenes** *(Datasource* = **Datos.EspecimenDS,** *Datafield* = **Nombre).**

El componente *TDBLookup ComboBox* obtiene los datos de la tabla **Especies** *(ListSource* = **Datos.EspeciesDS,** *ListField* = **Common_name,** *Key_Field* = **Category).** En dicho *combo* se muestra el campo *Common_name* de cada uno de los registros existentes en la **biolife.db.** Cuando el usuario selecciona un nombre de especie, lo que se guarda realmente en la tabla **Especimenes** es el campo *Category* de la tabla **Especies.**

Puesto que en este componente el usuario no puede introducir datos, nos aseguramos que siempre va a seleccionar una especie existente. El nombre de la especie que seleccionemos en el *combo* se almacena en el campo **Especie** de la tabla **Especimenes** (gracias a que *DataSource* = **Datos.EspecimenDS** y *DataField* = **Especie).**

El *TSpeedButton* que se encuentra a su derecha sirve para mostrar un nuevo formulario, donde aparecen las características de la especie seleccionada. No es necesario buscar la especie seleccionada en la tabla **Especies.** Puesto que el *TDBLookUp ComboBox* enlaza con esta tabla, al elegir una especie, realmente estamos posicionando el cursor sobre un determinado registro de la tabla **Especies,** cuyos datos se muestran en el nuevo formulario de forma automática.

Un TDBLookUp permite asegurar que el usuario introducirá un dato correcto

El *TDBGrid* muestra los datos de la tabla **Movimientos** *(Data Source* = **MovimientosDS).** En dicho *TDBGrid* no se pueden editar los datos directamente, sólo podemos visualizarlos. Para ello, hemos definido las propiedades adecuadas (por ejemplo, *ReadOnly* = **True).**

Un *TDBgrid* no debería usarse para la edición/introducción de datos. Siempre es preferible utilizar un formulario. La razón es que en el *TDBGrid* no podemos controlar al 100% todas las acciones efectuadas por el usuario. Por este motivo, personalmente no recomiendo el uso indiscriminado de los *TDBGrids* para la edición de datos.

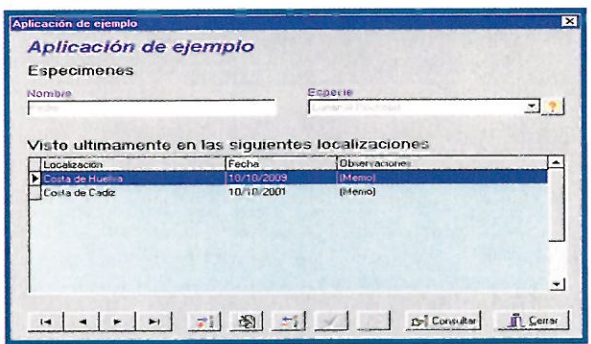


Figura 4.- Formulario principal de la aplicación.

Cuando el usuario quiera editar, introducir, o simplemente ver los datos de un movimiento con más comodidad, realizará un doble *click* sobre el *TDBGrid,* con lo que se mostrará un formulario de edición de datos.

El control *TDBNavigator* permite movernos por los registros de la tabla **Especimenes.** Por tanto, estará asociado al *TDataSource* **EspecimenDS.** Cuando cambiemos de registro activo, automáticamente cambiarán los datos que se muestran en el *TDBGrid* (gracias a la relación *master/detail* definida en el *TDataModule).*

Los botones permiten añadir, editar y borrar registros. En este caso, se ha optado por no permitir que el usuario introduzca/edite datos a menos que previamente haya pulsado el botón de insertar/editar.

No es conveniente la edición de datos en un TDBGrid

Existe una función, llamada *ActivarBotones,* que en función del estado de la tabla **Especimenes,** activará unos botones u otros. Por regla general, cuando se esté en modo de edición, sólo podremos actuar sobre los botones aceptar/cancelar. Cuando la tabla esté en modo de visualización, podremos actuar sobre todos los botones, excepto sobre estos dos.

El botón de insertar realiza un *Append* de la tabla **Especimenes** y, posteriormente, desactiva todos los botones, excepto los de aceptar/cancelar. El botón de edición hace lo mismo, excepto que la operación ahora es un *Edit.*

El botón de borrar elimina el registro activo de la tabla **Especimenes,** pero aquí hay que tener en cuenta un pequeño deta-

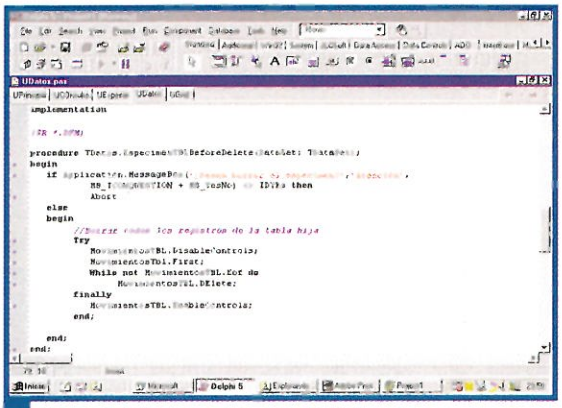


Figura 5.- Código asociado al botón de borrar.

lle. Cuando borramos un espécimen, tenemos que borrar además todos los movimientos que tuviese. Para realizar este borrado, aprovecharemos el evento *BeforeDelete* del componente *EspecimenesTBL*.

Al borrar un espécimen, se deben borrar sus movimientos

Antes de borrar el registro de la tabla **Especimenes,** recorremos la tabla **Movimientos,** eliminando todos los registros asociados al espécimen actual (ver Figura 5). Además, aprovechamos el *BeforeDelete* para preguntar al usuario si realmente desea borrar el espécimen. Si responde que no, abortaremos el borrado (lanzado un *Abort).*

Los botones **Aceptar/Cancelar** realizan un *Post/Cancel* de la tabla de **Especimenes.** Por último, el botón **Consultar** llama a un nuevo formulario que permite buscar un determinado espécimen en la base de datos.

FORMULARIO DE ESPECIES

Este formulario pertenece a la aplicación *biolife,* incluida entre las demos de *Delphi.* Simplemente he copiado la unidad, reorganizado los componentes y les he cambiado a todos la propiedad *DataSource,* de forma que ahora todos usan el *TDatasource EspecieDS* situado en el *TDa tamodule.*

Además, y puesto que no vamos a permitir que el usuario modifique ningún dato, todos los controles tienen su propiedad *ReadOnly* a **True** (y además, para distinguirlos de los controles «editables», se les ha cambiado el color de fondo).

INSERTAR DATOS EN EL TDBGRID

Como hemos comentado, no permitiremos al usuario editar datos directamente en el *TDBGrid* (no es nada recomendable). En su lugar, al hacer doble *click,* se mostrará el formulario que se observa en la Figura 6.

Los controles de edición de la parte superior (en color celeste) están enlazados con el *TDataSource* de la tabla **Especimenes.** En ellos se mostrarán los datos del espécimen que estemos editando/visualizando. Así el usuario sabe en cada instante con qué espécimen está tratando.

El resto de los campos están enlazados con el *TDataSource* de la tabla **movimientos.** Existen dos *TDBNavigator.* El de la izquierda permite movernos por los diferentes movimientos. Sólo está activo cuando estamos visualizando el espécimen. El de la derecha permite añadir nuevos movimientos o editar el actual. Sólo estará activo cuando estemos editando/insertando un nuevo espécimen.

Al botón de cerrar se le ha añadido el código necesario para que se cancele la edición si el usuario pulsa sobre él, y esos momentos se está editando o insertando un movimiento.

El evento BeforePost permite controlar si se han rellenado campos obligatorios

Por último, se ha añadido algo de código al evento *BeforePost* del componente **MovimientosTBL** (en el TDatamodule). Aprovechamos este evento para controlar que se han introducido los datos mínimos para poder dar de alta un movimiento, que en este caso son la fecha y la localización). El código es muy simple:

```
procedure
    TDatos.MovimientosTBLBefore
    Post
(DataSet: TDataSet);
begin
   if (MovimientosTBLLocalizacin.
       AsString='') or
       (MovimientosTBLFecha.
       AsString='')  then
```

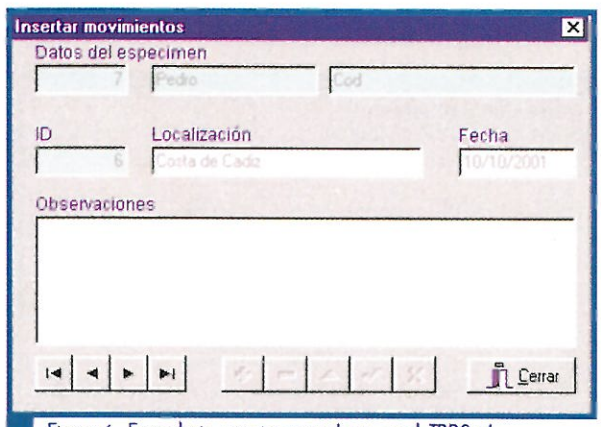


Figura 6.- Formulario para insertar datos en el *TDBGrid.*

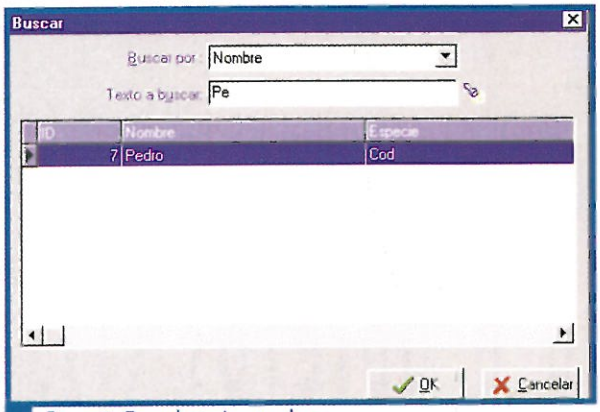


Figura 7.- Formulario de consultas.

```
    Begin
       ShowMessage('Faltan datos
      por
                  introducir');
       Abort;
    end;
end;
```

Si no se cumplen las condiciones, simplemente se lanza un *Abort*. Así evitamos que se introduzcan registros incompletos.

Además, hemos aprovechado el evento *OnSetText* del campo **Fecha,** para controlar que la fecha introducida sea correcta. En caso de error, se mostrará un mensaje al usuario, en perfecto castellano. Para acceder al evento *OnSetText,* hay que hacer doble *click* sobre **MovimientosTBL,** seleccionar el campo *Fecha,* y pinchar en la pestaña eventos del *Inspector de Objetos).* El código es el siguiente:

```
try
  StrToDate(Text);
  TField(Sender).AsString:=Text;
except
  ShowMessage('Valor de fecha no
    válido');
  Abort;
end;
```

Es importante que antes de empezar a añadir datos a la tabla **Movimientos** se hayan confirmado los cambios de la tabla **Especimenes.** Si no es así, los datos que insertemos no enlazarán con posterioridad con el espécimen en edición. Esto se realiza en el evento *OnDblClick* del *TDBgrid.* En general, antes de comenzar a editar en una tabla *detail,* se deben haber confirmado los cambios en la tabla *master.*

CONSULTANDO ESPECÍMENES

Al pulsar el botón **Consultar** (en el formulario principal), aparece la pantalla que se observa la Figura 7. En ella tenemos un componente *TComboBox* y un *TEdit.* El *TComboBox* permite seleccionar el campo por el que queremos buscar (en este caso, Nombre y Especie).

El método Locate permite hacer búsquedas en los DataSets

En el *TEdit* el usuario tecleará las primeras letras del nombre o especie que desea localizar. Al pulsar la tecla **Enter** se lanzará una consulta *SQL* (usando el componente *QAuxiliar del TDataModule).* Dicha consulta muestra sus resultados en el *TDBGrid* de la parte inferior. El código es el siguiente:

```
procedure TFConsulta.
SearchButtonClick(Sender:
    TObject);
Var buscar:String; CadSql:String;
    Valor:TDateTime;
begin
  Buscar:=SearchEd.Text;
  CadSql:='Select * from
    especimenes where'
     +CampoBusqueda.text+' like '
     +
      QuotedStr(Buscar+ '%') ;

    LanzarConsulta(datos.Auxiliar
    QR, CadSql);
end;
```

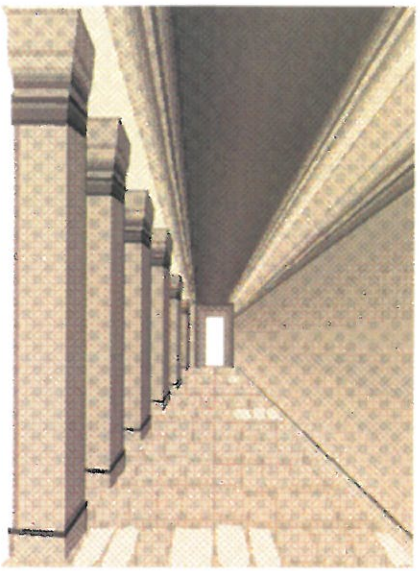
La función *LanzarConsulta* pasa la cadena *SQL* al *TQuery* y lo abre. Consulte el código en el *CD-ROM.* En próximos artículos se mostrarán algunos conceptos relacionados con el *SQL.*

Cuando el usuario localice el espécimen deseado, basta con pulsar el botón **OK** o hacer doble *click* en el *TDBGrid.* En dicho momento se tomarán los datos del registro seleccionado en el *TDBGrid,* que se usarán para lanzar una consulta sobre el componente **EspecimenesTBL** (usando el método *Locate).*

```
procedure TFConsulta.
BitBtn1Click(Sender: TObject);
begin
   if not
    datos.AuxiliarQR.IsEmpty
   then
       //nos movemos al especimen
       //indicado
       Datos.EspecimenesTBL.
       Locate('ID',

    datos.AuxiliarQR.Fields[0].
              AsString ,
              [loPartialKey]);
   ModalResult:=mrOk;
end;
```

El método *Locate* recibe como parámetro el campo (o campos) sobre los que se realizará la búsqueda, el valor (o valores) a buscar y por último si la búsqueda es parcial, sensitiva a mayúsculas, las dos, o ninguna de las dos.

Al cerrar la ventana de consulta, nos encontraremos editando el registro que habíamos seleccionado en el *TDBgrid.*



Programación orientada a objetos (I): novedades de VB 7

Jordi Agost Moré. *Profesor universitario.*

La nueva versión de *Visual Basic 7* incluye numerosas mejoras en el área de orientación a objetos y también modernas características adicionales propias de los lenguajes más avanzados.

INTRODUCCIÓN

Empezaremos esta pequeña serie comentando las novedades de *Visual Basic 7,* el paradigma de programación visual hacia donde nos encaminamos, para luego en posteriores entregas estudiar el punto de partida en el que nos encontramos actualmente y examinar las posibilidades que tenemos hasta la llegada de la nueva versión.

Hoy en día la mayor parte de las aplicaciones se orientan en mayor o menor medida hacia el mercado de *Internet*. Esto presupone que las aplicaciones deben ser cada día más escalables, robustas y sobre todo reutilizables.

Estas son características fácilmente asimilables a los lenguajes de programación orientados a objetos. Utilizando este tipo de lenguajes se logran con una relativa facilidad aplicaciones altamente escalables, fáciles de actualizar, simples de mantener y sobre todo de depurar.

En los últimos años *Visual Basic* ha ocupado un sector muy significativo dentro del conjunto de lenguajes de desarrollo rápido de aplicaciones *Windows*. Sin embargo, la falta de características de orientación a objetos ha limitado su penetración en el mercado para crear componentes y aplicaciones basadas en la arquitectura de tres capas, utilizada para realizar aplicaciones destinadas a *Internet*.

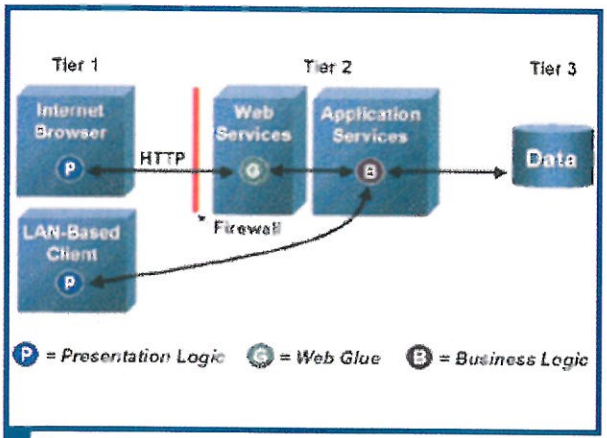


Figura 1.- Imagen de una arquitectura de tres capas.

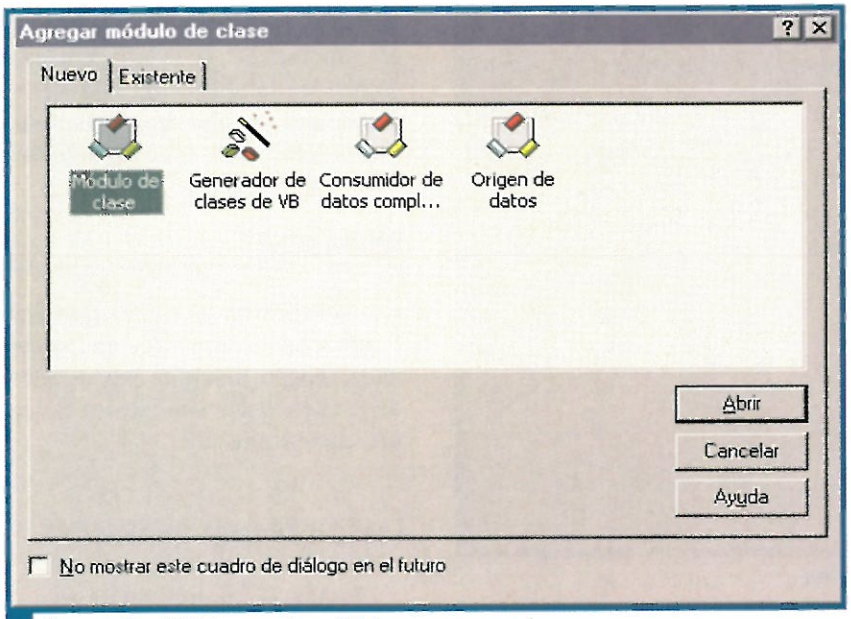


Figura 2.- Posibilidades que ofrece *VB* al crear una nueva clase.

Para paliar esta deficiencia, las novedades de la próxima generación de *Visual Basic* se han centrado precisamente en este aspecto, a fin de lograr maximizar y simplificar el desarrollo de aplicaciones *Web*. Se pretende con ello que *Visual Basic* como lenguaje adquiera la potencia de *C++* o *Java* sin por ello perder la facilidad de desarrollo de la que siempre ha hecho gala.

La falta de características de orientación a objetos en VB ha limitado la posibilidad de crear aplicaciones de tres capas

Las principales novedades en las características de orientación a objetos del lenguaje son:

- Herencia.
- Polimorfismo.
- Sobrecarga de operadores.
- Encapsulación.
- Constructores.

Y en lo que se refiere a mejoras generales del lenguaje, las más significativas son:

- Multihilo.
- Excepción de errores.
- Tipos seguros.
- Miembros compartidos.
- Inicializadores.

MEJORAS EN LA POO

Vamos a ver en primer lugar las mejoras en el ámbito de la programación orientada a objetos.

HERENCIA

En una programación orientada a objetos la herencia sería el hecho de compartir los atributos y las operaciones entre distintas clases, tomando como base una relación jerárquica entre un grupo de ellas.

Sin duda, ésta ha sido con diferencia una de las novedades más demandadas por los usuarios de *Visual Basic*. Ahora finalmente verán satisfechas sus demandas a través de la instrucción *Inherits*.

Imaginemos una clase:

```
ClassA
    Function PruebaA()
    ...
    End Function
```

Una clase llamada B podría heredar los atributos y procedimientos de A y, además tener sus propios atributos y procedimientos:

```
ClassB
    Inherits ClassA
    Function PruebaB
    ...
    End Function
```

Con la sentencia Inherits podremos aplicar la herencia de métodos e interfaces en nuestras clases

La sentencia *Inherits* alberga diversas características. En primer lugar, la clase derivada puede soportar todos los métodos e interfaces soportados por la clase base (en anteriores versiones de *Visual Basic* existía una herencia bastante primitiva que sólo se limitaba a la interfaz).

En segundo lugar, la clase derivada puede autorizar a las clases derivadas a redefinir los métodos definidos en la clase base de la cual derivamos. Esto lo realizará a través de la palabra *Overrides*. Además, como observábamos en este último ejemplo, la clase derivada puede ampliar si lo desea el conjunto de métodos e interfaces soportados por la clase base (en este último caso se extendía con la función *PruebaB)*.

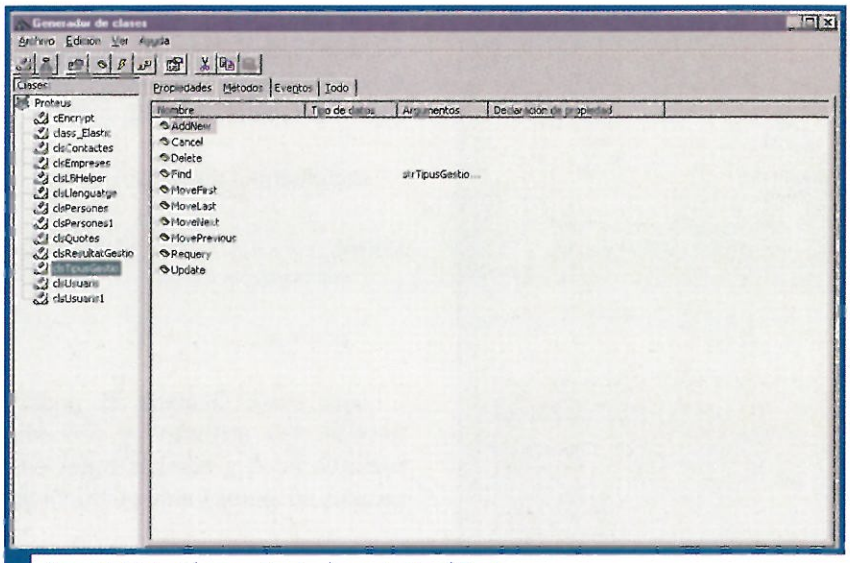


Figura 3.- Vista del generador de clases de *Visual Basic*.

POLIFORMISMO

El polimorfismo es la posibilidad que incorpora un lenguaje con el fin de procesar los objetos de forma distinta dependiendo de su tipo o clase. También permite redefinir los métodos para las clases derivadas mediante la herencia.

El polimorfismo permite redefinir los métodos para las clases derivadas

Por ejemplo imaginemos que estamos realizando un programa sobre el control de comisiones en un banco, dentro del cual existe una clase genérica llamada *Cliente*. Dicha clase tiene una función implementada que se llama *CalculoComision*, que podría calcular (de forma extremadamente simplificada) el tiempo que aplicamos la comisión por la variable que aplicamos:

```
Class Cliente
    Function CalculoComision
     CalculoComision = Tiempo *
     Comision
    End Functions
```

Con la nueva estructura Try...Catch...Finally se simplificará el tratamiento de errores en Visual Basic

Pero supongamos que de esta clase derivamos ahora, por ejemplo, la clase *Empleado*. Al ser los empleados *Clientes* preferentes tendrán una comisión más baja. Entonces el polimorfismo permitirá definir diferentes tipos de métodos *"CalculoComision"* en distintas clases derivadas, por ejemplo:

```
Class Empleado
    Inherits Cliente
    Function CalculoComision
     CalculoComision = Tiempo *
    Comision * Descuento
    End Functions
```

No importa en qué subclase estemos haciendo el cálculo, ya que el método *CalculoComision* devolverá en cada momento el resultado correcto. Por lo tanto, en función de la clase utilizada dentro de la aplicación *(Cliente* o *Empleado)* se aplicará el método *CalculoComision* correspondiente a cada una de ellas.

SOBRECARGA

La sobrecarga permite que los métodos y operadores de los objetos tengan distintos significados dependiendo del contexto en el que nos encontremos.

Con la sobrecarga los métodos y operadores tendrán distintos significados dependiendo de los elementos que estemos utilizando

Sería algo parecido a lo que ocurría con el operador + de *Visual Basic,* pero ampliado al conjunto de todos los operadores. Dicho operador se utiliza tradicionalmente para realizar la operación de suma, pero también puede utilizarse para concatenar cadenas. Así:

```
5 + 4
```

es una expresión válida, pero también lo es:

```
"hola " + "mundo"
```

Los operadores tendrán distintos significados dependiendo del tipo de datos o clases con las que estén operando. Así **objeto1 + objeto2** pueden realizar diferentes operaciones dependiendo de la naturaleza de *objeto1* y *objeto2*. Por ejemplo imaginemos que deseamos sumar números, pero también deseamos sumar dos objetos de tipo *variant,* y también dos imágenes. Tradicionalmente hubiésemos utilizado tres funciones

con nombres distintos como por ejemplo:

```
Sub SumaNumeros ( intTmp1 As
     Integer, intTmp2 As Integer)

Sub SumaArrays (obj1 As Variant,
     obj2 As Variant)
Sub SumaImagenes (Img1 As Object,
     Img2 As Object)
```

Ahora con la sobrecarga podemos utilizar el mismo nombre para las tres operaciones, incluyendo la sentencia *Overloads*.

```
Overloads Sub Suma ( intTmp1 As
     Integer, intTmp2 As Integer)

Overloads Sub Suma (obj1 As
     Variant, obj2 As Variant)
Overloads Sub Suma (Img1 As
     Object, Img2 As Object)
```

ENCAPSULACIÓN

El hecho de que un lenguaje orientado a objetos posea encapsulación significa que los programadores pueden ocultar, si lo desean, información de un objeto. Nos estamos refiriendo a información como, por ejemplo, estructuras de datos y código. Dicho procedimiento es una forma de aislar la complejidad de un objeto del resto de la operación.

A través de la encapsulación garantizamos el estado de un objeto ya que tan sólo él puede acceder a los datos que hemos marcado como encapsulados. Como consecuencia el mantenimiento del código resulta mucho más fácil, ya que si existe un problema en el acceso a datos, sabremos inmediatamente dónde se encuentra dicho problema.

Con la encapsulación sólo un objeto puede acceder a sus datos

En la nueva versión de *Visual Basic* podremos implementar la encapsulación a través de la palabra

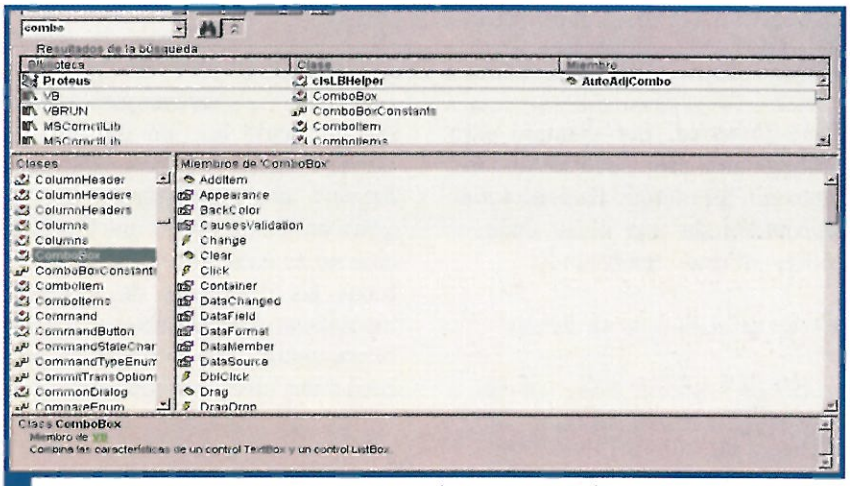


Figura 4.- Examinador de *VB* donde vemos cómo los objetos que utilizamos internamente son tan sólo clases.

http://www.demasiado.com

¡camos **programadores** web para unirse a los que hacen posible la comunidad de Demasiado en Internet.

ofrece:

emuneración atractiva en uno de los proyectos en plataforma Linux más grandes de España.
ontrato indefinido.
bertad para vestir como se quiera.

requiere:

uen rollo: gran capacidad de desarrollo creativo en equipo.
xperiencia en Php, Perl, Java, C, C++ y/o Delphi.
mplios conocimientos de SQL.
- Disponibilidad para trabajar en Madrid.
- Desconocimiento total de FrontPage.

íbenos indicando tu historial profesional, la referencia del puesto y el medio preferido de contacto a:
rrhh@demasiado.com
Garantizamos absoluta confidencialidad en el proceso de selección.

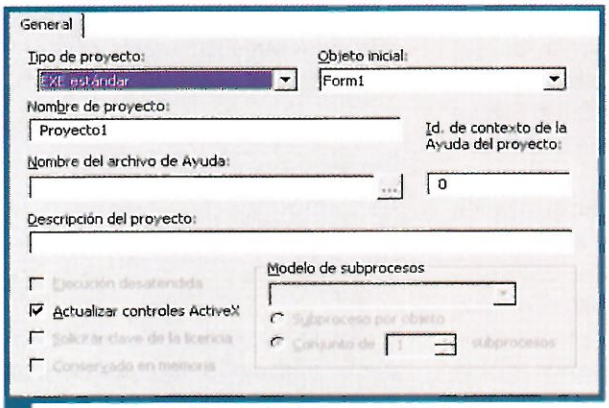


Figura 5.- Posibilidades multihilo (sección "Modelo de subprocesos") de *VB 6*, limitadas a proyectos *DLL ActiveX* o *EXE ActiveX* sin interfaz gráfica.

clave *Protected*. Por ejemplo para proteger una propiedad *strNombre*, junto con un método llamado *CambioNombre* de una clase dado el código utilizado tendríamos:

```
Protected strNombre As String

Protected Function CambioNombre()
    Me.strNombre = NewName
End Function
```

MEJORAS DEL LENGUAJE

En este punto repasaremos brevemente las mejoras generales que introduce la nueva versión de *Visual Basic:*

- Multihilo.
- Excepción de errores.
- Tipos Seguros.
- Miembros compartidos.
- Inicializadores.

MULTIHILO

Hoy en día la mayor parte de las aplicaciones escritas con *Visual Basic* son síncronas. Esto significa que cada línea de código debe ser ejecutada antes de que se ejecute la siguiente línea (aunque esto sucede sobre todo en las aplicaciones gráficas, ya que en algunos casos como las aplicaciones *DLL ActiveX* y *EXE ActiveX* que no incorporen elementos gráficos pueden incorporarse técnicas asíncronas).

Como la mayor parte de las aplicaciones que se están desarrollando hoy en día van en mayor o menor medida orientadas a *Internet,* nos encontramos con un grave problema, que es que la aplicación no es escalable. Para satisfacer todas las demandas de nuestros hipotéticos clientes debemos utilizar herramientas que permitan un proceso o una ejecución concurrente.

Con la inclusión de capacidades multihilo podremos ejecutar dentro de nuestra aplicación tareas en un segundo plano

Con la inclusión de las características multihilo en nuestra aplicación, ésta va a ser capaz de ejecutar una rutina, una consulta o la función que deseemos en segundo plano mientras nuestra aplicación sigue el curso normal, obteniendo así un procesamiento asíncrono. Una nueva sentencia, o mejor dicho objeto, se ha añadido para tal efecto, se trata de la orden *Thread.*

Imaginemos que tenemos una clase llamada *Prueba* y en ella un método llamado *Trabajo:*

```
Class Prueba
    Sub Trabajo
     ...
    End Sub
End Class
```

Y ahora imaginemos que desde el hilo o núcleo central de nuestro programa deseamos lanzar otro hilo donde se ejecute dicha subrutina. El código dentro de la nueva versión de *Visual Basic* sería el siguiente:

```
Dim clsTmp As Prueba
Dim ThrTmp As Thread
Set clsTmp = New BackGroundWork()
Set ThrTmp = New Thread (New
    ThreadStart(AddressOf
    ThrTmp.Start)
```

Donde *ThrTmp* sería un nuevo hilo de la aplicación.

EXCEPCIONES DE ERRORES

Tradicionalmente uno de los puntos débiles de *Visual Basic* ha sido su pobre manejo de todo el sistema de errores. Hasta hoy, los programadores de *Visual Basic* tenían que proveer líneas de control de errores en cada subrutina. Existían, no obstante, diversas soluciones comerciales (por ejemplo *vbGuard)* que instalándose como ampliacio-

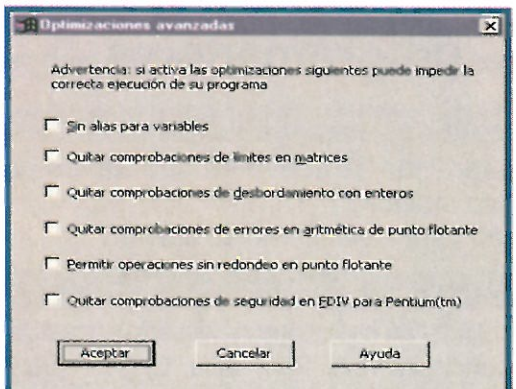


Figura 6.- Opciones de compilador, donde observamos las posibilidades entre adaptación de tipos.

nes dentro del área de trabajo de *Visual Basic* realizaban de una forma más o menos automática el control de errores y el siempre engorroso trabajo de poner todas las etiquetas e instrucciones pertinentes para su control.

La política de control de errores se realizaba con la instrucción *On Error GoTo*. Esto debido a la complejidad de código resultante dificultaba el mantenimiento de las aplicaciones. Hasta ahora cuando ocurría un error, el control o flujo del programa era transferido a una etiqueta que trataba el error y seguidamente se abandonaba la rutina (mediante la orden *Exit Sub)* o bien se continuaba en el flujo que previamente se había abandonado (con las órdenes *Resume* y *Resume Next).*

Con ello cuando nos encontramos con diversos tratamientos por distintos errores el código puede volverse completamente ilegible ya que se entra en una serie de idas y venidas a través del mismo y de las diferentes etiquetas de control de errores.

Para evitar esto, se ha procedido a añadir una estructura que controla las posibles excepciones que puedan existir en el código y se ha provisto al lenguaje con una estructura y un código que se ejecuta igualmente en condiciones de funcionamiento normal y de error. Dicho paso se logra con la instrucción *Try...Catch...Finally:*

```
Try
 'código de ejecución normal
Catch
 'código de excepción
Finally
 'Acción final
End Try
```

TIPOS SEGUROS

Hasta la última versión *Visual Basic* el programador tenía total libertad a la hora de generar los tipos implícitos de variables. Cuando asignamos valores o pasamos parámetros sin indicar un tipo explícitamente, el compilador de *Visual Basic* intenta ajustar en tiempo de ejecución la variable pasándola a un tipo que pueda contenerlo (sería el típico hecho que asignamos a una variable numérica, el contenido de una caja de texto sin convertir el valor mediante la función *Val).*

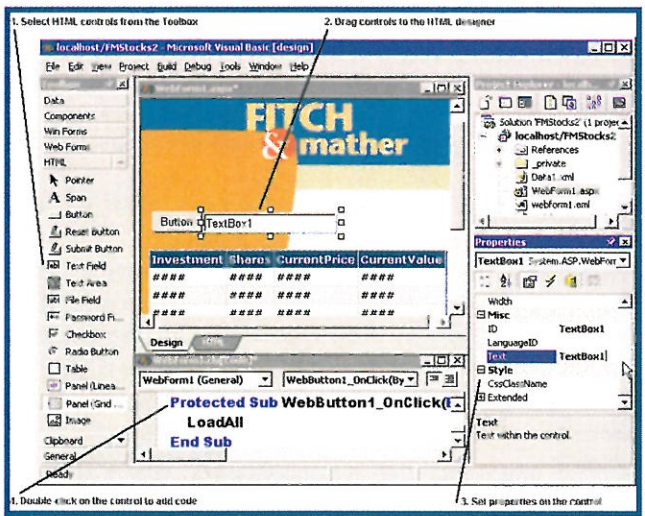


Figura 7.- Los *webforms* son una de las grandes novedades que presenta *Visual Basic 7.*

Dichos ajustes en tiempo de ejecución pueden hacer que el programa falle cuando se ejecuta, en el caso de que el valor que hemos pasado no pueda ser convertido sin pérdida de datos.

Para evitar este problema se ha añadido una nueva posibilidad dentro de las opciones de compilación, a través de la orden *Option Strict.* Dicha opción generará los errores en tiempo de compilación para cada conversión que podría llegar a fallar.

INICIALIZADORES

Otra de las nuevas características del lenguaje de la nueva versión de *Visual Basic* son los inicializadores. Esto significa que las variables van a poder inicializarse en la misma línea en que son declaradas y dicha inicialización va a poder realizarse en cualquier punto del código incluyendo el interior de las estructuras de control.

Con la nueva versión de VB se puede inicializar las variables en el mismo momento de declararlas

Así, el código que hasta ahora escribíamos de esta forma:

```
Dim strTmp As String
StrTmp = "hola"
```

Ahora lo podremos escribir:

```
Dim strTmp As String = "hola"
```

Dicho avance aumentará de forma significativa la modularización y legibilidad del código.

CONCLUSIÓN

Hemos visto un breve resumen de las principales novedades que incorporará la nueva versión de *Visual Basic.* En próximos artículos analizaremos las posibilidades de orientación a objetos que ofrece la actual versión 6.0, centrándonos en lo más significativo: las clases.

Visual C++ y MFC (III)

Constantino Sánchez Ballesteros.
Técnico Superior Dearrollo Aplicaciones Informáticas.

Después de presentar los conceptos fundamentales de la programación con *Visual C++*, entraremos en detalles abordando una técnica muy utilizada: la subclasificación. Con ella lograremos, por ejemplo, controlar la información tecleada dentro de las cajas de texto incluidas en las ventanas.

SUBCLASIFICACIÓN

Cuando insertamos una caja de texto en un formulario (ventana), podemos ver que al ejecutar la aplicación sólo tenemos la funcionalidad de teclear caracteres dentro de la caja, y como mucho, limitar su número de entrada.

Mediante este funcionamiento resulta imposible interceptar, por ejemplo, la tecla **Intro** o los caracteres que se van introduciendo. Esto es debido a que el control caja de texto, cuando se inserta en una ventana desde el editor de recursos no tiene un mapa de mensajes como lo tiene su ventana padre.

Para resolver esta limitación se utiliza la subclasificación. Cuando una aplicación crea una ventana, el sistema operativo le asigna un bloque de memoria para almacenar información específica sobre la misma, incluyendo la dirección del procedimiento de ventana que procesa los mensajes para dicha ventana. Cuando el sistema operativo necesita pasar un mensaje a la ventana utiliza la dirección del procedimiento de ventana para buscar información específica de la ventana y pasar el mensaje a ese procedimiento.

La subclasificación permite a cualquier aplicación interceptar y procesar mensajes enviados o pasados a una ventana particular antes de que la ventana pueda procesarlos. Al subclasificar una ventana, nuestra aplicación podrá tener un mayor control sobre su comportamiento. No es conveniente subclasificar ventanas, aunque podamos hacerlo, pero esta técnica sí es útil para controles como las cajas de texto o cuadros de lista.

En el caso de la ventana, cuando la subclasificamos, se reemplaza la dirección de procedimiento de la misma con la dirección del nuevo procedimiento subclasificado. Cuando este procedimiento subclasificado reciba un mensaje, se podrán efectuar tres acciones:

- Pasar el mensaje al procedimiento original.
- Modificar el mensaje y pasarlo al procedimiento original.
- Procesar el mensaje y no pasarlo al procedimiento original.

Figura 1.- Paquete de desarrollo de *Microsoft*.

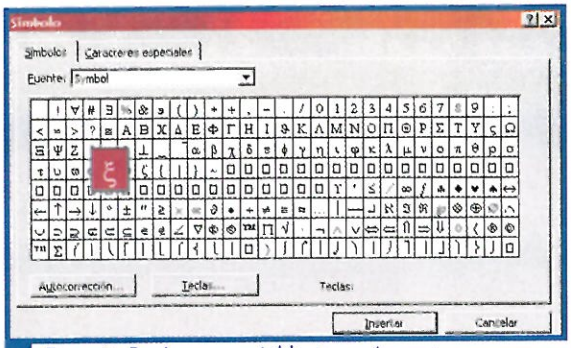


Figura 2.- Puede ser muy útil limitar qué caracteres se introducen en una caja de texto.

TECLADO

Cuando pulsamos teclas, el gestor del teclado envía a *Windows* los caracteres correspondientes. Hay que tener en cuenta que la pulsación de una tecla conlleva dos procesos distintos:

1. Cuando se pulsa la tecla (*Keydown*).
2. Cuando se suelta la tecla pulsada (*Keyup*).

Dentro del entorno *Windows* se distinguen dos tipos de pulsaciones:

1. Sistema: se generan cuando se utiliza una combinación específica de teclas. En concreto, la tecla **Alt+[otra tecla]**.
2. No de sistema: resto de teclas.

Recordemos que la tecla **Alt** es utilizada comúnmente para la activación de menús u otras operaciones especiales (copiar, pegar, etc.) de *Windows*.

Cuando se pulsa o se suelta una tecla, *Windows* coloca en la cola de mensajes de la aplicación activa tanto los mensajes de sistema como los restantes. En la tabla 1 se puede apreciar un ejemplo.

Cuando se escribe un caracter por medio de la pulsación de una tecla el proceso que sigue *Windows* es el siguiente:

- Envía el mensaje *WM_KEYDOWN* (tecla virtual pulsada).
- Envía el mensaje *WM_CHAR* (Código *ASCII* de la tecla pulsada).
- Envía el mensaje *WM_KEYUP* (tecla virtual que se pulsó).

Windows posee un teclado virtual en el que se especifican las teclas que puede recibir. Las teclas virtuales más utilizadas están definidas dentro del archivo **windows.h**. En la tabla 2 se definen varias de estas teclas.

INTERCEPTACIÓN DE TECLAS PULSADAS

Como hemos visto anteriormente, cuando se pulsan teclas el sistema operativo envía los mensajes correspondientes a la ventana que ha generado las pulsaciones. Para poder interceptar cualquier tecla pulsada (o combinaciones) que no produzcan código *ASCII*, se utilizarán los mensajes *WM_KEYDOWN* y *WM_KEYUP*.

Cuando se pulsa una tecla, Windows envía un mensaje al programa que generó la pulsación

En *C++* la función que se encarga de la manipulación de estos mensajes utiliza los siguientes argumentos:

- *NChar*: contiene el código de la tecla virtual correspondiente a la tecla pulsada.
- *NRepCnt*: contiene el número de repeticiones que se producen cuando el usuario mantiene pulsada la tecla
- *NFlags*: contiene el estado de la tecla, el cual puede contener información como la que se describe en la Tabla 3.

La función comentada anteriormente tiene el siguiente aspecto:

```
void CMiEdit::OnX(UINT nChar,
    UINT nRepCnt, UINT nFlags) {
  // TODO: Add your message
  CEdit::OnKeyUp(nChar, nRepCnt,
    nFlags);
}
```

TABLA 1. Ejemplo de pulsación de teclas.

	MENSAJES NO SISTEMA	MENSAJES DE SISTEMA
TECLA QUE SE PULSA	WM_KEYDOWN	WM_SYSKEYDOWN
TECLA QUE SE SUELTA	WM_KEYUP	WM_SYSKEYUP

TABLA 2. Teclas virtuales.

TECLA	TECLA VIRTUAL (WINDOWS)
Intro	VK_RETURN
ESC	VK_ESCAPE
SHIFT	VK_SHIFT
CONTROL	VK_CONTROL

TABLA 3. Información de Nflags.

BITS	DESCRIPCIÓN
0 a 7	Código del teclado
8	Indicador de tecla extendida: 1 = tecla extendida 0 = no
13	Código de contexto: 1 = tecla Alt pulsada 0 = no
14	Estado previo de tecla: 1 = pulsada 0=suelta.
15	Estado de transición: 1 = la tecla se suelta 2 = la tecla se pulsa

CMiEdit correspondería a una clase creada por nosotros para establecer la posterior subclasificación. Más adelante ya veremos cómo hacerlo. *OnX* representa la función que va a gestionar un mensaje de teclado determinado. A continuación se muestran algunos ejemplos:

- *OnKeyDown*: cuando hay que procesar un mensaje *WM_KEYDOWN*.
- *OnChar*: cuando hay que procesar un mensaje *WM_CHAR*.
- *OnKeyUp*: cuando hay que procesar un mensaje *WM_KEYUP*.

El código *nChar* devuelto por esta función no diferencia entre mayúsculas o minúsculas. Para ello tendríamos que utilizar *OnChar*.

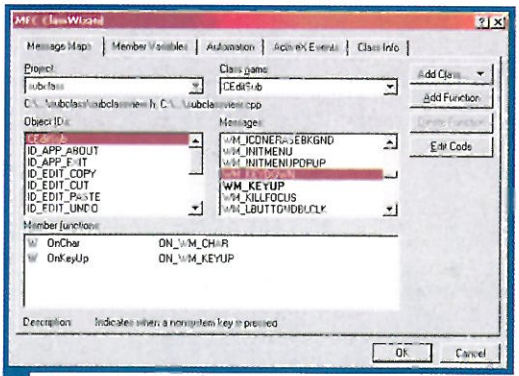


Figura 3.- En *Visual C++* podemos interceptar teclas mediante funciones.

Para que en esta función podamos deducir si se pulsó mayúsculas o no, debemos ver si el argumento *nChar* es *VK_SHIFT* y después analizar si tiene un carácter de la **A** a la **Z**. En el siguiente ejemplo podemos ver cómo detectar si se ha pulsado la tecla **Z** mayúscula o minúscula:

```
void CMiEdit::OnKeyDown(UINT
    nChar,
    UINT nRepCnt, UINT nFlags) {
  if (nChar==VK_SHIFT)
     mayusculas=true; //SE PULSÓ
    SHIFT
  if (nChar=='Z' && mayusculas)
     TRACE("Z\n"); // SE PULSÓ Z
    MAYÚSCULA
      if (nChar=='Z' &&
¡mayusculas)
      TRACE("z\n"); // SE
PULSÓ z MINÚSCULA
  CEdit::OnKeyUp(nChar,
nRepCnt, nFlags);
}

void CMiEdit::OnKeyUp(UINT
nChar,
   UINT nRepCnt, UINT
nFlags) {
  if (nChar==VK_SHIFT)
     mayusculas=false;
//SE SOLTÓ SHIFT
  CEdit::OnKeyUp(nChar, nRepCnt,
    nFlags);
}
```

El control CEdit no tiene mapa de mensajes si se inserta desde el Editor de recursos

La variable **mayúsculas** indicará si se ha pulsado la tecla **Shift** o no. La macro *TRACE* visualiza los argumentos especificados en la ventana de salida *Debug*. Esto es muy útil en procesos de depuración.

En el siguiente ejemplo podemos ver cómo cancelar un carácter si no se encuentra dentro del rango permitido por el usuario. En concreto, prohibiremos la entrada de caracteres a aquellos campos que no sean numéricos:

```
void CMiEdit::OnChar(UINT nChar,
   UINT nRepCnt, UINT nFlags) {
  if (nChar<'0' || nChar>'9')
    MessageBeep(0xFFFFFFFF); //no
    se pulsó un número
  else
    CEdit::OnKeyUp(nChar,
    nRepCnt, nFlags);
}
```

La función *MessageBeep* permitirá emitir un sonido por el altavoz del *PC* cuando no se tecleen caracteres numéricos.

PROYECTO DE EJEMPLO

A continuación vamos a desarrollar un ejemplo con lo explicado hasta ahora. En concreto se pretende crear una ventana con un cuadro

de lista y una simple caja de texto en la cual, cuando pulsemos la tecla **INTRO** se pasarán los datos contenidos en la misma al cuadro de lista. Además, limitaremos la entrada de caracteres numéricos, con lo cual, cuando se introduzca un número se generará un sonido por el altavoz y no se escribirá dicho carácter.

Los mensajes WM_CHAR permiten controlar qué caracteres se han pulsado

Si no subclasificáramos la caja de texto, al pulsar **INTRO** sobre la misma no podríamos controlar dicho suceso, es decir, no ocurriría nada puesto que este control no permite el mapeado de mensajes si se ha creado mediante el editor de recursos.

Para la creación del proyecto seguiremos los pasos que hemos visto en entregas anteriores, es decir, crearemos una aplicación con el asistente que permita el soporte *Documento/Vista*. Una vez creado el proyecto insertaremos los controles correspondientes desde el editor de recursos. Para ello, seleccionaremos **ResourceView** en la pestaña de la ventana del *Workspace*.

En el contenedor de la ventana insertaremos un control *EditBox* (caja de texto/edición) y una *ListBox* (cuadro de lista). Una vez insertados, seleccionaremos la caja de texto y activaremos el asistente de clases (*ClassWizard*) mediante la pulsación simultánea de las teclas **CTRL+W**. Nuestro objetivo es crear una clase de tipo *CEdit* para poder realizar la subclasificación.

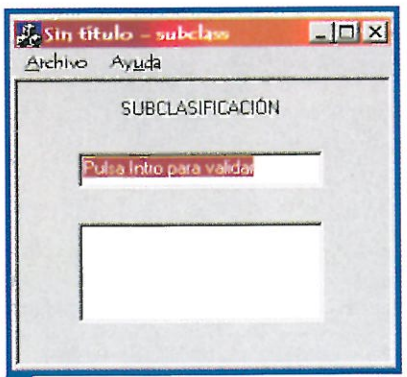


Figura 4.- Ejemplo creado en este artículo.

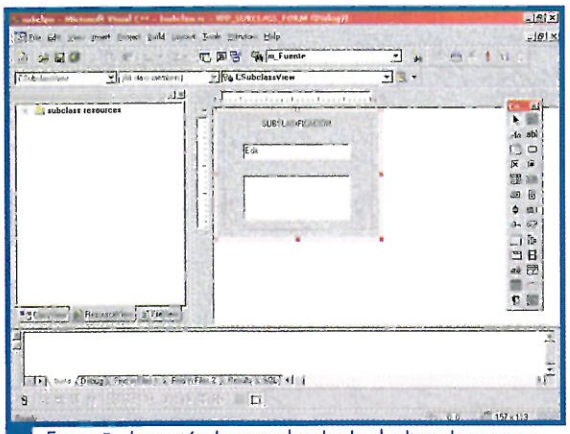


Figura 5.- Inserción de controles desde el editor de recursos.

Es necesario derivar de *CEdit* ya que buscamos subclasificar un control de este tipo. Debemos pulsar sobre **Add Class** para que aparezca una pantalla en la que generaremos la clase, que denominaremos **CEdit Sub** y provocaremos que se genere dentro de los archivos creados para la vista de nuestro programa (*SubclassView.h* y *SubclassView.cpp*).

Para poder subclasificar un control insertado en la vista del programa es necesario que la clase creada (*CEditSub*) se declare antes que la clase de la vista (*CSubclassView*). Esto es debido a que al subclasificar llamaremos a la nueva clase creada desde la clase de la vista, *CSubclassView*, y si *CEditSub* no se ha declarado con anterioridad, la clase no se podrá reconocer.

Esta operación es muy sencilla, aunque tendremos que hacerla manualmente. Consiste en cortar del archivo de cabecera de la vista la declaración de la clase *CEditSub* y pegarla antes de la declaración de la clase de la vista (*CSubclassView*). Después de seguir estos pasos, el resultado quedará como sigue:

- Declaración de clase para subclasificar:

```
class CEditSub : public CEdit {
// Construction
public:
    CEditSub();
...
...
...
};
```

- Declaración de clase para la Vista:

```
class CSubclassView : public
    CFormView {
  protected: // create from
    serialization only
CSubclassView();
DECLARE_DYNCREATE(CSubclassView)

...
...
}
```

Hecha esta operación ya podemos subclasificar. Para ello volveremos a entrar en el *ClassWizard* y seleccionaremos la pestaña *Member Variables*. A continuación tomaremos la clase *CSubclassView* y pulsaremos sobre el identificador *IDC_EDIT1*, siendo este punto determinante en todo el proceso. Para ello, pulsaremos sobre el botón **Add Variable** y crearemos una variable del siguiente tipo:

- Nombre: **m_Objeto**
- Categoría: **Control**
- Tipo: **CEditSub**

A continuación provocamos que el control *IDC_EDIT1* se base

TABLA 4. Ayuda del método SetSel.

```
CListBox::AddString

int AddString( LPCTSTR lpszItem );

VALOR DE RETORNO

LB_ERR si ocurrió un error; LB_ERRSPACE si no hay espacio
suficiente para guardar la nueva cadena de caracteres.

PARÁMETROS

lpszItem

puntero a una cadena de caracteres que se agregará al control.

NOTAS
Esta función se utiliza para agregar una cadena de caracteres
a un cuadro de lista. Si el cuadro de lista no se creó con el
estilo LBS_SORT, el texto se agregará al final de la lista. En
caso contrario, la cadena se insertará en la lista y ésta se
ordenará automáticamente.
```

ahora en la nueva clase que hemos creado (*CEditsub*).

Ya sólo queda crear los procedimientos que controlarán el comportamiento de la caja de texto. Como se comentó anteriormente, queremos que al pulsar la tecla **Intro** se pase el contenido de la caja al cuadro de lista. Además, si se pulsan números, se generará un mensaje de aviso y no se escribirá ningún caracter.

Vayamos por partes. Para controlar la pulsación de la tecla **Intro** debemos activar la función que gestiona el mensaje *WM_KEYUP*. También es posible realizar la misma tarea utilizando WM_KEYDOWN.

Para generar la función utilizaremos el *ClassWizard*. Desde la pestaña *Message Maps* (mapas de mensajes) seleccionaremos en el apartado *object ID* la clase *CEditSub* y en el apartado *Messages* seleccionaremos *WM_KEYUP*. Hecho esto pulsaremos sobre el botón **Add Function** y seguidamente sobre el botón **Edit Code** para comenzar a programar el evento.

El Listado 1 incluido en el *CD-ROM* muestra las operaciones a realizar.

En el procedimiento se crean las variables **caja** y **lista** para obtener punteros a los objetos insertados en el formulario.

El siguiente paso consiste en tomar el contenido de la caja de texto dentro de la variable **texto** para que, posteriormente podamos insertarlo en el cuadro de lista. Después de insertar el texto en la lista (método *AddString*) tenemos que borrar el contenido de la caja de texto para la siguiente entrada de datos. Por supuesto, este paso no es requerido pero permite una mayor funcionalidad del control y un mejor aspecto visual.

Para proceder al borrado es necesario seleccionar todo su contenido. Esto se hace utilizando el método *SetSel*. Con él podemos seleccionar parte o todo el contenido del control. Necesitaremos establecer el primer parámetro con el valor **0** y el segundo con **–1**. En la Tabla 4 se muestra la ayuda para utilizar el método *SetSel*.

Una vez seleccionado el texto, sólo nos queda asignar al contenido un espacio en blanco y ya habremos logrado limpiar la caja de texto.

CEditsub es la nueva clase de tipo CEdit creada para la subclasificación

Para finalizar con la aplicación, queda comprobar si se introducen números en la caja de texto. Para ello debemos interceptar el mensaje *WM_CHAR*. Una vez más procederemos a activar *ClassWizard* y añadiremos el mensaje *WM_CHAR* del mismo modo que hicimos con *WM_KEYUP*. Al editar el procedimiento quedará el código como sigue:

```
void CEditSub::OnChar(UINT nChar,
    UINT nRepCnt, UINT nFlags) {
```

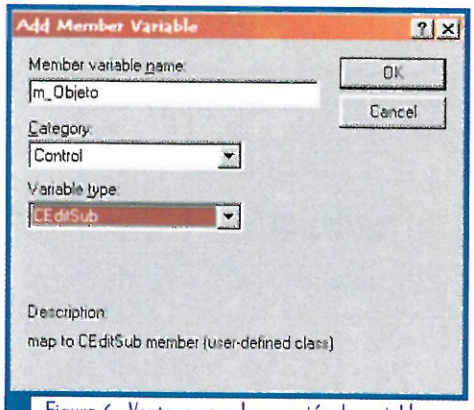


Figura 6.- Ventana para la creación de variables miembro.

```
  if (nChar>='0' && nChar<='9')
    MessageBeep(MB_ICONEXCLAMA-
    TION);
  else
    CEdit::OnChar(nChar, nRepCnt,
    nFlags);
}
```

El caracter introducido por teclado es devuelto por la variable **nChar**. Si éste está dentro del rango de los números produciremos un sonido de alerta por el altavoz. En caso contrario, retornaremos el caracter al control para que se visualice.

Como punto final, podemos hacer que la ventana se visualice en el centro de la pantalla y aparezca un mensaje por defecto en la caja de texto (por aquello de la estética). Para ello, debemos utilizar el método *CenterWindow* dentro del procedimiento *OnInitialUpdate* correspondiente a la vista.

```
void
    CSubclassView::OnInitialUpdat
    e() {
  CFormView::OnInitialUpdate();
  GetParentFrame()-
   >RecalcLayout();
  ResizeParentToFit();
  GetParent()->CenterWindow ();
  SetDlgItemText
   (IDC_EDIT1,"Pulsa Intro para
    validar");
}
```

En la próxima entrega aprenderemos a utilizar los menús, incluyendo su creación, entradas de menú, etc.

TABLA 5. Edición de nChar.

```
CEdit::SetSel

void SetSel( int nStartChar, int nEndChar);

PARÁMETROS:

nStartChar

Especifica la posición de comienzo para la selección. Si
nStartChar es 0 y nEndChar es -1, se seleccionará todo el
texto del control. Si nStartChar es -1, se eliminará la selec-
ción actual.

nEndChar

Especifica la posición final.
```


CON MEDIALAB BUSCA A
LOS MEJORES PROGRAMADORES

RES el mejor en lo que haces, pero quieres más. Quieres estar
nde innovar sea parte del día a día. Te entusiasmaría colaborar en
futuro de las mejores compañías del mundo.
uieres llevar los medios digitales un paso más allá.

OMOS la consultora líder en comunicación digital. Asesoramos a
uestros clientes en el diseño de su estrategia de futuro en internet.
stamos a la vanguardia en comunicación estratégica y en las
cnologías más avanzadas.
ontamos con los mejores profesionales integrados en una sólida
d internacional.

E ofrecemos la oportunidad de trabajar para las principales
mpañías en los más avanzados proyectos en internet.

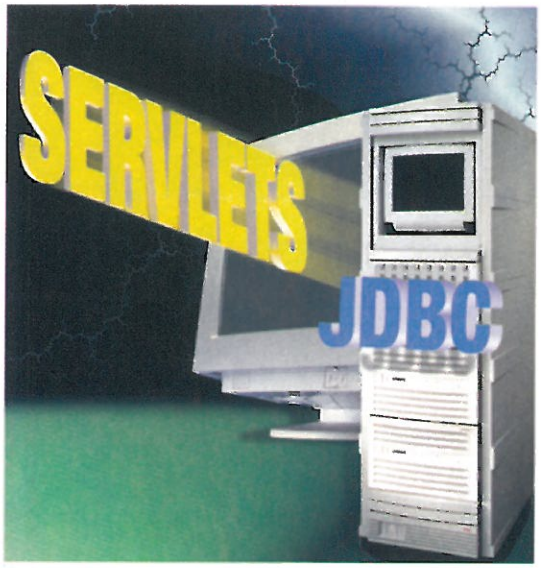
star HOY donde los demás estarán en el futuro.

Todo lo que debes hacer es ir a
www.iconmedialab.es/jobs
y contarnos dónde quieres estar en los próximos años.

También puedes enviarnos tu solicitud por correo a:
Icon Medialab España
C/Alcalá, 21-8o Dcha.
28014 Madrid

Icon Medialab tiene oficinas en Estocolmo, Bruselas,
San Francisco, Londres, París, Hamburgo, Milán, Madrid,
Helsinki, Tampere, Copenhague, Oslo y Kuala Lumpur.

ICON MEDIALAB



Aplicaciones Web con acceso a BBDD basadas en Java (IV)

Adolfo Aladro García.
Analista Programador

La gestión de las conexiones que se establecen con la base de datos desde el *servlet* es uno de los puntos críticos de este tipo de aplicaciones. Por ello, a lo largo de este capítulo estudiaremos algunas de las distintas configuraciones que pueden tomarse para que nuestra aplicación *Web* funcione de la manera más eficiente posible.

GESTIÓN DE LAS CONEXIONES

Uno de los factores más importantes que deben ser tomados en consideración a la hora de plantear una aplicación *Web* basada en *servlets* y *JDBC* es aquel que tiene que ver directamente con cómo y cuándo se establecen las conexiones con las bases de datos desde el *servlet*. Conviene en este punto recordar el ejemplo que ilustraba el capítulo anterior de esta serie. El cuerpo central del *servlet* contenía las siguientes líneas:

```
String urlBD = "jdbc:odbc:COLEC-
    CIONDISCOS";
Class.forName("sun.jdbc.odbc.Jdb-
    cOdbcDriver");
Connection conexion = Driver
    Manager.getConnection(urlBD);
...
conexion.close();
```

Cada vez que el servidor recibe una petición contra ese *servlet* se crea un nuevo *thread* y el cuerpo central del procedimiento *service* se ejecuta ahí. Esto significa que por cada petición se abre y se cierra una conexión con la base de datos. Este comporta-

miento es, por ejemplo, muy ineficiente para aplicaciones *Web* en las que el número de peticiones se mantiene constante, y relativamente alto, durante casi todo el tiempo.

La gestión de las conexiones resulta de gran importancia

Una de las ventajas que proporcionan los *servlets* es el hecho de que se ejecutan en *threads*, lo que implica que pueden compartir fácilmente información, tal y como adelantamos al principio de esta serie. La flexibilidad, unida a la potencia de *Java*, ofrece multitud de configuraciones distintas a la hora de gestionar las conexiones con la base de datos. A continuación se expondrán algunas de estas posibles soluciones, señalando sus ventajas y sus incovenientes, pero en cualquier caso no se puede concluir taxativamente que una es mejor que otra ya que depende de muchos factores: las caracteríticas y los requisitos de la propia aplicación, el tráfico que se espera, la configuración de la base de datos, etc. Cada caso particular requiere un análisis pormenorizado.

UNA SOLA CONEXIÓN

Como se vió anteriormente, todo *servlet* tiene un ciclo de vida predeterminado. Hasta el momento todos los ejemplos que hemos desarrollado utilizaban principalmente el método *service*, que es ejecutado por los *threads* del *servlet* de forma concurrente. Sin embargo, existen otros dos métodos que también pueden ser sobreescritos y que forman parte del ciclo de vida. Nos referimos a los métodos *init* y *destroy*.

Los threads pueden compartir una conexión

El método *init* es invocado únicamente cuando el *servlet* se carga al principio. Esto puede ocurrir la primera vez que se realiza una petición contra él, o bien cuando el servidor arranca si así se ha configurado. Este método recibe como único parámetro un objeto de tipo *ServletConfig*. Éste permite acceder a los parámetros de inicialización que hayan sido establecidos al configurar el servidor o el motor de ejecución de *servlets*. También permite el acceso al contexto del *servlet* a través de un objeto del tipo *ServletContext*.

El método *destroy* se ejecuta cuando el servidor no necesita el *servlet*, o el *servlet* va a ser cargado de nuevo. Se encarga de liberar todos los recursos que consume el *servlet*.

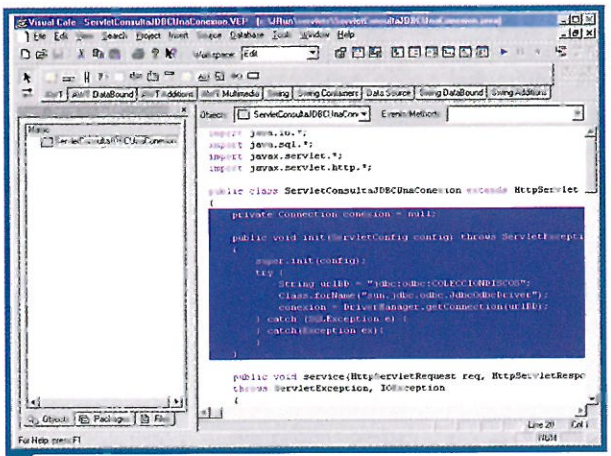


Figura 1.- Código fuente del *Servlet* que utiliza una sola conexión para atender a todas las peticiones.

Además de lo anterior debemos recordar que un *servlet* no es más que una clase y como tal puede tener sus propiedades. Éstas tienen la particularidad de ser recursos compartidos por todos los *threads* del *Servlet*. Así es posible crear una variable del tipo *Connection* que pertenezca a la clase.

```
public class ServletConsultaJDB-
    CUnaConexion
      extends HttpServlet {
  private Connection conexion =
    null;
  ...
}
```

Esta variable representa la conexión a través de la cual se desarrollarán todas las consultas a la base de datos. Para que esto sea posible el método *init* tiene que presentar el siguiente aspecto:

```
public void init(ServletConfig
    config)
     throws ServletException {
  super.init(config);
  ...
}
```

Init se usa para inicializar variables compartidas

En primer lugar se llama al método *init* de la superclase de la que depende el *servlet* para que ésta se responsabilice del tratamiento del parámetro aplicando el comportamiento por defecto que tenga establecido.

El siguiente paso consiste en establecer la conexión con la base de datos. Para ello creamos la dirección *URL JDBC*:

```
String urlBD = "jdbc:odbc:
    COLECCIONDISCOS";
```

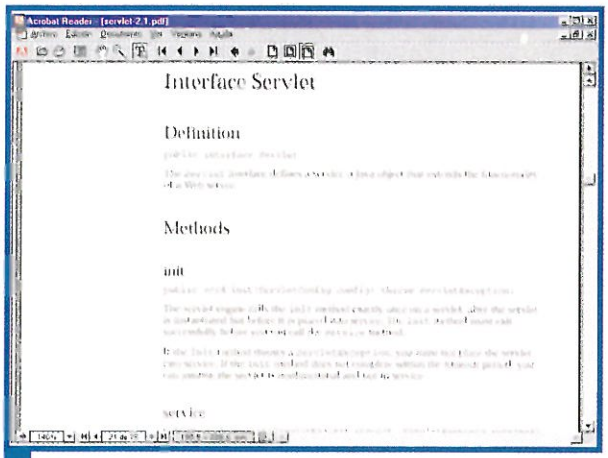


Figura 2.- Documentación de la *API Java Servlets* donde se detallan las características del método *init* de la clase genérica *Servlet*.

Cargamos la clase que implementa el controlador, tal y como hacíamos en los ejemplos del capítulo anterior:

```
Class.forName("sun.jdbc.odbc.JdbcOdbcDriver");
```

Por último, la conexión se establece utilizando el método *getConnection* de la clase *DriverManager*.

```
conexion = DriverManager.getConnection(urlBD);
```

A diferencia de lo que ocurría en los ejemplos que se han estudiado hasta el momento, estas líneas de código solamente se ejecutan una vez al cargarse el *servlet*. La variable **conexión** es compartida por todas las peticiones que acepta el *servlet* y que ejecuta concurrentemente en *threads* independientes.

La conexión establecida con la base de datos debe ser cerrada únicamente cuando el *servlet* muera, evento que viene asociado a la ejecución del método *destroy*.

```
public void destroy() {
  try {
    if (conexion != null) conexion.close();
  } catch (SQLException e) { }
}
```

Tal y como se puede observar en las líneas precedentes, el método *destroy* comprueba si la variable **conexión** tiene un valor distinto de **null**. En caso afirmativo significa que hay una conexión abierta y entonces la cierra.

VENTAJAS E INCONVENIENTES

La solución que acaba de ser planteada presenta sus ventajas e inconvenientes. Por un lado se ahorra en recursos. No debemos olvidar que toda nueva conexión con la base de datos consume memoria, ancho de banda, etc. Sin embargo la utilización de una sola conexión no representa una opción válida para muchos casos.

No debemos olvidar que se deben liberar los recursos utilizados

Imaginemos por un momento que implementamos un sistema transaccional en el que se debe garantizar que se ejecutan todas las consultas de un conjunto o ninguna de ellas. Con una única conexión, compartida por todos los usuarios que realizan peticiones simultáneamente, sería imposible llevar a cabo un correcto control del sistema. ¿En qué orden llegan las peticiones? ¿Cuándo sabríamos el momento de ejecutar una sentencia *commit* en la base de datos para salvar los cambios producidos por las consultas?

UNA LISTA DE CONEXIONES

La utilización de una lista de conexiones es una de las alternativas más atractivas que soluciona los problemas planteados por la opción descrita anteriormente y que al mismo tiempo ofrece un mecanismo mucho más eficiente que el mero hecho de abrir y cerrar una conexión por cada petición. Para ello podemos crear una clase nueva, que llamaremos *ListaConexiones*, y que se encargará de gestionar las conexiones que se establezcan con la base de datos.

LA CLASE LISTACONEXIONES

Un objeto de tipo *Hashtable* representa una tabla de información donde cada elemento tiene un valor (*value*) y se diferencia de los demás por su clave (*key*). Como clave puede utilizarse cualquier objeto que sea no nulo. Por lo tanto podemos crear una lista de conexiones utilizando un objeto de tipo *Hashtable*. Los elementos clave serán los objetos de tipo *Connection* devueltos por el método *getConnection* de la clase *DriverManager*. Cada conexión tendrá además asociado un valor *booleano* que será **true** si la conexión está siendo utilizada, o **false** en otro caso.

Se puede gestionar una lista de conexiones

La lista de conexiones será dinámica en el sentido de que se añadirán nuevas conexiones en el caso de que se requieran y no haya ninguna disponible. Los objetos de tipo *HashTable* pueden crearse sin tamaño definido y éste crece o decrece dinámicamente a medida que se insertan o eliminan entradas.

El archivo **ListaConexiones.java** contiene la implementación de la clase que representa a la lista de conexiones y que será utilizada por el *servlet*.

El constructor de la clase *ListaConexiones* recibe dos parámetros:

la dirección *URL JDBC* de la base de datos (*urlJDBC*) y la clase correspondiente al controlador *JDBC* que ha de utilizarse. Para los ejemplos de esta serie estos dos datos son suficientes, pero si fueran necesarios usuario y contraseña para acceder a la base de datos, habría que añadirlos.

Se abren tantas conexiones como sea necesario

La variable privada *conexiones* es de tipo *Hashtable* y almacena, como es evidente, la tabla de las conexiones. Además tenemos también dos variables privadas que son *urlJDBC* y *N_CON_INICIALES* que guardan respetivamente la dirección *URL JDBC* de la base de datos y el número inicial de conexiones que serán abiertas.

En primer lugar asignamos las propiedades *urlJDBC* y *conexiones*:

```
this.urlJDBC = urlJDBC;
conexiones = new Hashtable();
```

El siguiente paso consiste en cargar la clase correspondiente al controlador *JDBC* que permite acceder a esa base de datos.

```
Class.forName(controlador);
```

Por último el constructor se encarga de añadir a la lista tantas conexiones como indique la variable *N_CON_INICIALES*:

```
for(int i=0; i<N_CON_INICIALES;
    i++) {

    conexiones.put(DriverMana-
    ger.getConnection(urlJDBC),

    Boolean.FALSE);
}
```

El método *put* de la clase *Hashtable* añade un nuevo elemento a la tabla de información. El primer parámetro que recibe se corresponde con la clave y el segundo con el valor. Ninguno de ellos puede ser nulo. El valor asociado a todas las conexiones que se abren inicialmente es *false* porque todas están libres para ser utilizadas por aquellos *threads* del *Servlet* que así lo soliciten.

La clase *ListaConexiones* cuenta con dos métodos:

- *obtenerConexion*
- *devolverConexion*

EL MÉTODO OBTENERCONEXION

Este método será llamado por un *thread* del *servlet* cada vez que se desee obtener una conexión para ejecutar una sentencia *SQL* a través de ella. Debe recorrer la lista de conexiones en busca de una que esté libre. Si la encuentra, la devuelve y la marca como ocupada. En otro caso debe crear una nueva conexión y añadirla a la lista.

Hay que vigilar el acceso simultáneo a variables compartidas

En primer lugar debemos ser conscientes de que la lista de conexiones será un objeto utilizado por distintos *threads* del *servlet* al mismo tiempo. Esto significa que tenemos un recurso compartido al que se accede de manera concurrente, o lo que es lo mismo, que tenemos que poner los medios necesarios para evitar situaciones de inconsistencia como la que refleja la Figura 4.

La clase ListaConexiones gestiona las conexiones

Java proporciona un mecanismo de exclusión mútua muy sencillo. La palabra reservada *synchronized* garantiza que dos o más *threads* no podrán ejecutar simultáneamente el mismo bloque de código. Estas líneas de código constituyen así lo que se denomina como sección crítica en el ámbito de la programación concurrente.

```
synchronized (conexiones) {
...
}
```

Siempre que se establece una sección crítica es necesario observar una serie de puntos básicos:

- A fin de mantener la metodología de orientación a objetos, las secciones críticas deben establecerse preferiblemente mediante la utilización de métodos declarados como *synchronized* en vez de selec-

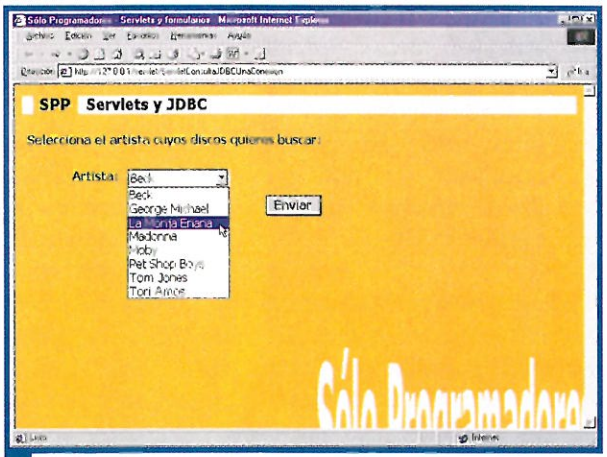


Figura 3.- Resultado que proporciona el *Servlet* ServletConsultaJDBCUnaConexión.java

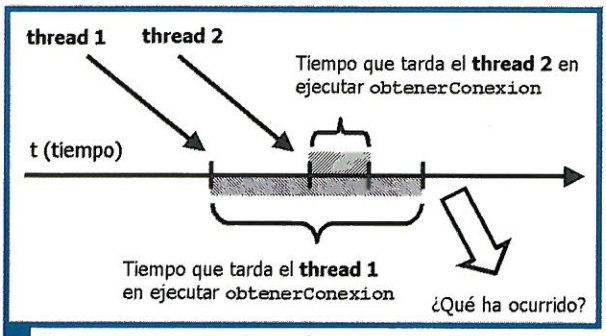


Figura 4.- Esquema que ilustra el caso típico que puede producirse al permitir que dos *threads* ejecuten simultáneamente una sección crítica.

cionar tan sólo unas líneas dentro de un método.

- Las secciones críticas deben ser lo más pequeñas posibles de forma que los mecanismos de protección no degraden el funcionamiento del sistema. Como se puede apreciar en la Figura 5, la introducción de secciones críticas suele conllevar la aparición de tiempos de espera.

- Deben evitarse los fenómenos de hambruna: un *thread* se queda esperando indefinidamente para acceder a la sección crítica ya que siempre hay algún otro que se adelanta; e interbloqueo: los *threads* de un conjunto de *threads* están esperando para acceder a la sección crítica por otro *thread* del mismo conjunto, el cual no puede avanzar a su vez porque también está esperando. El resultado es que ningun *thread* consigue entrar en la sección crítica.

El método *keys* de la clase *Hashtable* devuelve un objeto de tipo *Enumeration* con todas las claves de la tabla de información, en otras palabras, con todas las conexiones guardadas en la lista de conexiones. La interfaz *Enumeration* sirve para mantener colecciones de elementos. Los métodos *hasMoreElements* y *nextElement* se utilizan para recorrer la colección. El primero devuelve *false* si se han recorrido todos los elementos o *true* en otro caso. El segundo permite acceder a cada uno de los elementos de la colección.

```
for (Enumeration cs =
     conexiones.keys();cons.
     hasMoreElements();) {
  c = (Connection)cs.next
     Element();
  ...
}
```

El bucle anterior permite recorrer toda la colección de conexiones devueltas por el método *keys*.

Tenemos que comprobar el valor asociado para cada conexión. Si éste es *false* significa que la conexión está libre y por lo tanto puede utilizarse. El método *get* de la clase *Hashtable* recibe como parámetro una clave y devuelve el valor asociado en la tabla de información.

```
Boolean ocupada = (Boolean)
     conexiones.get(c);
if (ocupada == Boolean.FALSE) {
  ...
}
```

Si encontramos una conexión libre simplemente debemos marcarla como ocupada y devolverla. El método *put* de la clase *Hashtable* permite reasignar un valor a una clave dada (si la clave no existe introduce el nuevo par clave/valor tal y como hemos explicado en el código del constructor de la clase).

```
conexiones.put(c, Boolean.TRUE);
return c;
```

En el caso de que no haya conexiones disponibles será preciso aumentar la lista de conexiones. Esta operación se encuentra al margen de la sección crítica ya que no se precisa garantizar la exclusión mútua para crear nuevas conexiones.

```
conexiones.put(DriverManager.get
     Connection(urlJDBC),
     Boolean.FALSE);
```

Una vez que se ha creado una nueva conexión, el método *obtenerConexion* se llama a sí mismo, ya que en realidad lo que hay que hacer es de nuevo lo mismo.

Las secciones críticas deben ser cortas

Solamente queda pendiente un detalle. Las conexiones pueden perderse por cualquier motivo (la configuración de la propia base de datos con respecto a los *timeouts*, un fallo ocasional de la red; etc.) por lo que puede ocurrir que una conexión tenga el valor asociado como *false* y que sin embargo no esté disponible. Por ello, antes de asignarla debemos cercionarnos de que efectivamente la conexión está todavía funcionando correctamente. Esto puede realizarse de diversas formas dependiendo del grado de seguridad que se desee mantener. Por ejemplo, podríamos hacer lo siguiente:

```
Boolean cerrada = Boolean.FALSE;
Boolean error = Boolean.FALSE;
try {
  cerrada = c.isClosed();
} catch(SQLException e) {
  error = Boolean.TRUE;
  c = DriverManager.get
    Connection(dbURL);
}
if (cerrada == Boolean.TRUE &&
     error == Boolean.FALSE) {
```

```
    c = DriverManager.get
      Connection(dbURL);
}
```

EL MÉTODO DEVOLVERCONEXION

Este método recibe como único parámetro la conexión que ha de marcarse de nuevo como libre. Lo único que hay que hacer es buscarla en la lista de conexiones y asignarle el valor *false*. Para ello volvemos a recorrer la colección de conexiones devuelta por el método *keys* de la clase *Hashtable* y comparamos cada una de ellas con el parámetro *conexion_devuelta*.

```
if (c == conexion_devuelta) {
  conexiones.put(c,
    Boolean.FALSE);
  break;
}
```

La lista de conexiones es una variable compartida

En cuanto la encontremos, la marcamos como libre e interrumpimos la búsqueda.

EL SERVLET QUE USA LA CLASE LISTACONEXIONES

El *Servlet* que utiliza la clase que acabamos de describir es muy parecido al que hemos visto para el caso en el que se utiliza una sola conexión. En primer lugar debemos tener una variable que será común a todos los *threads* y que por lo tanto se declarará como propiedad de la clase:

```
public class ServletListaConexio-
    nes extends HttpServlet {
  private ListaConexiones lista-
    conexiones;
  ...
}
```

El método *init* deberá instanciar un objeto de la clase *ListaConexiones* para que todos los *threads* puedan utilizarlo:

```
...
String urlBD = "jdbc:odbc:
    COLECCIONDISCOS";
String controlador =
    "sun.jdbc.odbc.
    JdbcOdbcDriver";
listaconexiones = new Lista
    Conexiones(urlBD, controla-
    dor);
...
```

El cuerpo central del *Servlet* será similar salvo cuando se obtiene una conexión:

```
Connection conexion =
    listaconexiones.obtener
    Conexion();
```

y cuando se libera:

```
listaconexiones.devolverConexion(c
                onexion);
```

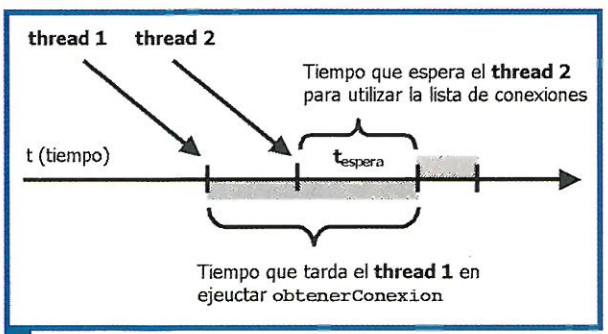


Figura 5.- Solución al problema de la sección crítica que consiste en la utilización de tiempos de espera.

Cuando se utiliza el método *devolverConexion* ésta se marca como disponible, pero en realidad hasta el momento no hemos hablado en ningún momento del cierre de las conexiones establecidas. Para solucionar esto se podría crear un nuevo método en la clase *ListaConexiones* que se denominará *cerrarLista* y cuyo código quedaría como sigue:

```
public void cerrarLista() throws
    SQLException {
  Connection c;
  for (Enumeration cs =
    conexiones.keys();cs.has
    MoreElements();) {
    c = (Connection)cs.
    nextElement();
    c.close();
  }
}
```

Se deben cerrar las conexiones antes de que muera el Servlet

Este método sería invocado en el código del método *destroy* del *Servlet* para liberar definitivamente todos los recursos.

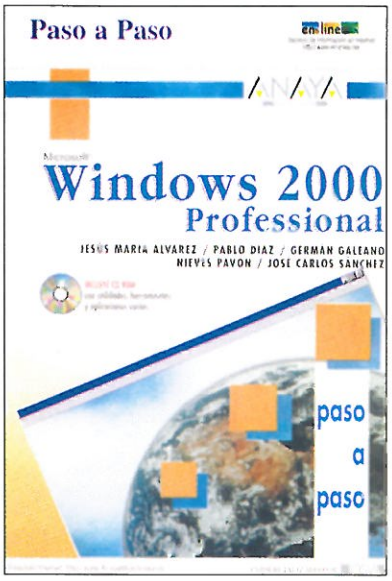
VENTAJAS E INCOVENIENTES

Este método reune varias ventajas. Por un lado, cada petición es atendida a través de su propia conexión por lo que pueden abordarse sistemas transaccionales o cualquier otro donde importa el orden en el que se atienden las consultas lanzadas contra la base de datos desde los diversos *threads* del *Servlet*. Pero al mismo tiempo se ahorran recursos ya que se evita tener que cerrar y abrir una conexión cada vez. Esto solamente se llevará a cabo durante un tiempo inicial de funcionamiento de la aplicación, hasta que la lista de conexiones alcance una extensión estable.

La principal contrapartida de este sistema es que puede llegar a darse el caso de tener muchas conexiones abiertas con las base de datos cuando en realidad únicamente se utilizan unas pocas.

WINDOWS 2000 PROFESSIONAL

Desde *Microsoft* dicen que *Windows 2000* es el sistema operativo del nuevo milenio. Una herramienta para trabajar en entornos profesionales aunque es capaz de adaptarse al entorno del usuario doméstico. Con el manual que nos presenta *Anaya* conseguirás aprender a manejarlo de una forma sencilla, paso a paso, con todos sus trucos.



Siguiendo sus indicaciones, la nueva versión de *Windows NT*, con cambio de nombre incluido, recuerda al *Windows* tradicional con su manejo de simples ventanas, iconos y botones hasta la personalización completa del escritorio, la administración del sistema o la seguridad.

Encontraremos explicaciones sobre el sistema de archivos, el explorador de *Windows*, aplicaciones, opciones de configuración, multimedia, navegación con *Internet Explorer*...

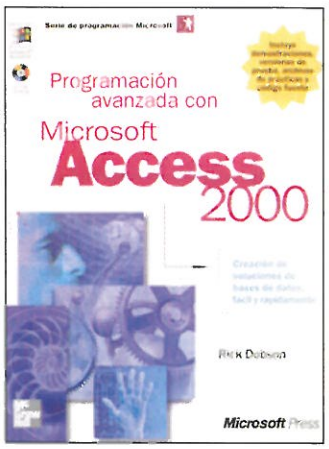
Dirigido tanto a usuarios básicos como medios, incluye un completo *CD-ROM* con utilidades a las que se les puede sacar mucho provecho.

Editorial: Anaya Multimedia
Número de páginas: 448
Niveles: Básico y Medio

Autores: J. M. Álvarez y otros
Idioma: Español
Precio: 3.995 Ptas. (I.V.A inc.)

PROGRAMACIÓN AVANZADA CON MICROSOFT ACCESS 2000

Los programadores que estén empleando *Access* 2000 ya pueden realizar personalizaciones sofisticadas gracias al manual que nos presenta *McGraw-Hill*. Con la ayuda de múltiples ejemplos, reunidos en un completo *CD-ROM* se realiza un aprendizaje más completo y detallado permitiendo el desarrollo de aplicaciones profesionales.



Gracias al soporte de *OLE DB* y de objetos de datos *ActiveX* contenidos en *Microsoft Access 2000* explicados en este libro se pueden obtener los datos con mayor rapidez y precisión. En poco tiempo nos encontraremos trabajando con conjuntos de registros, diseñando y construyendo tablas y consultas con ejemplos basados en *ADO*.

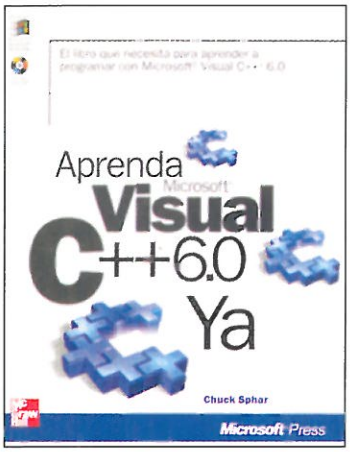
Además, esta guía avanzada nos permitirá integrar *Access 2000* con las bases de datos de *Microsoft SQL Server*, con una interfaz sencilla de manejar. Utilizaremos modelos de objeto de *Microsoft Office* y el lenguaje de programación *Visual Basic* para Aplicaciones (*VBA*) con el objetivo de acelerar las operaciones a realizar.

Y todo ello con la posibilidad de extender nuestras soluciones de datos para *Web*, incluso programando la seguridad por usuarios.

Editorial: McGraw Hill
Nº de páginas: 537
Nivel: Intermedio - Avanzado

Autor: Rick Dobson
Idioma: Español
Precio: 6.300 Ptas. (I.V.A inc.)

APRENDA MICROSOFT VISUAL C++ 6.0 Ya.



Aprender de una manera completamente didáctica y sencilla a programar en *Visual C++ 6.0* es lo que nos ofrecen con este libro.

Los que ya conozcan este lenguaje de programación podrán agregar complejas funcionalidades del programa con *MFC* avanzado y técnicas gráficas, mejorando sus habilidades. Para los principiantes lo mejor son las lecciones y los programas de ejemplo que nos ayudan a comprender el código necesario.

Este completo manual también nos ofrece la posibilidad de utilizar las *Microsoft Foundation Classes* (*MFC*) para construir aplicaciones basadas en *Win32*. La práctica del código *AppWizard* de *MFC* nos permitirá una mayor fluidez a la hora de programar, sobre todo gracias a los elementos que nos enseñarán a trabajar en el entorno de *C++* y familiarizarnos con él, con las herramientas, punteros, clases y objetos.

Además del tutorial de ayuda y el texto, el libro viene acompañado de un *CD-ROM* que incluye variados ejemplos de práctica y bibliotecas *MFC*.

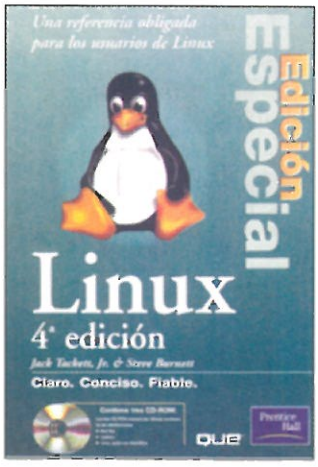
Editorial: McGraw-Hill
Número de páginas: 598
Nivel: Principiante

Autor: Chuck Spar
Idioma: Español
Precio: 6.500 Ptas. (I.V.A. inc.)

LINUX (4.ª edición). EDICIÓN ESPECIAL

Linux está de moda, de eso no cabe duda. Por eso las editoriales no dejan de sacar nuevos títulos al respecto. En este caso se trata de la 4ª edición de todo un clásico, con el que están demostrando que *Linux* ha dejado de ser el sistema operativo para una minoría.

Este libro incluye información actualizada del *RPM* (el Gestor de Paquetes de *RedHat*), con su sencillo modo de instalación y actualización del sistema. Por supuesto contamos con *PAM*, la utilidad de seguridad de casi todas las distribuciones *Linux*, así como de *X-Windows*.



En este ejemplar se incluye la mejor *suite* ofimática para *Linux*, *StartOffice*, con un completo paquete de programas de procesamiento de texto, hoja de cálculo, presentación y comunicaciones. Encontraremos además todas las opciones necesarias para instalar los distintos servicios (*DNS, Sendmail* y las *Usenet*) para *Internet* y configurar el servidor *Web* más utilizado en todo el mundo: *Apache*. Aprendiendo, además, a gestionar cuentas de correo de usuario y a configurar una red *TCP/IP*.

El libro se acompaña de tres *CD-ROM* con las distribuciones *Caldera* y *RedHat*, utilizadas en el manual y la suite *StarOffice*, junto con todo lo necesario para comprender de manera interactiva los conceptos de *Linux*.

Editorial: Prentice Hall
Nº de páginas: 1.069
Nivel: Intermedio

Autores: J. Tackerr, Jr. & S. Burnett
Idioma: Español
Precio: 7.995 Ptas. (I.V.A. inc.)

Dudas técnicas

En esta sección, como cada mes, SÓLO PROGRAMADORES os brinda la oportunidad de encontrar respuesta a las dudas que podáis tener. Ya sabéis que nuestra dirección es solop@virtualsw.es

PREGUNTA

Estoy creando una pequeña base de datos con *Oracle* y tengo muchas dudas. Las voy a resumir todo lo que pueda. Espero que podáis contestarme a algunas de ellas.

- Me han dicho que lo mejor que puedo hacer es que los campos que sean clave de las tablas sean de tipo *NUMBER* y que se correspondan con secuencias. ¿Qué son las secuencias? ¿Cómo se hacen? ¿Cómo hago para que un campo de una tabla apunte a una secuencia?

- He oído que dependiendo de cómo se escriban las *queries* a la base de datos, éstas pueden ser más rápidas o más lentas. ¿Podríais contarme algo de esto?

- ¿Qué diferencias existen entre los tipos *VARCHAR* y *VARCHAR2*? Porque yo no sé cuál de los dos escoger para meter campos de texto.

- ¿Cómo puedo sacar la fecha actual con *SQL* para meterla en una tabla mía? Las fechas en la base de datos, ¿con qué formato se guardan? ¿Puedo cambiarlo?

Muchas gracias y seguid así.

Vicente Soroño.

RESPUESTA

Estimado Vicente:

Vamos a intentar ayudarte con tus dudas paso a paso.

- Secuencias: Son un recurso excelente para mantener los identificadores de las tablas. Entre otras cosas, garantizan que no se dará el mismo identificador a los elementos de la tabla. Este caso puede producirse cuando el acceso a la base de datos es concurrente, como por ejemplo sucede en las aplicaciones de *Internet*.

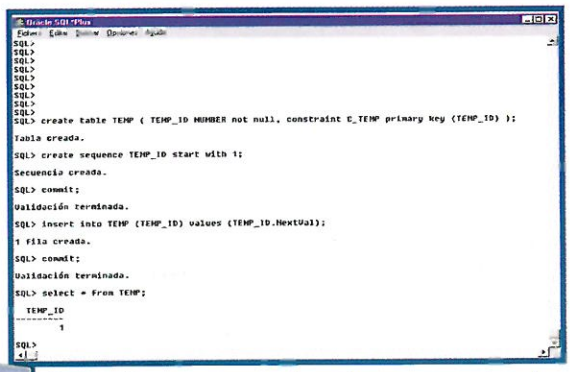


Figura 1.- Las secuencias pueden utilizarse para gestionar los identificadores de las tablas.

Supongamos que creamos una tabla de la siguiente forma:

```
create table TEMP ( TEMP_ID
    NUMBER not null, constraint
    C_TEMP primary key (TEMP_ID)
    );
```

El campo *TEMP_ID* es de tipo *NUMBER* y además es la clave de la tabla. Podemos crear una secuencia con el mismo nombre:

```
create sequence TEMP_ID start
    with 1;
```

La forma en la que se debe utilizar la secuencia es la que sigue:

```
insert into TEMP (TEMP_ID)
values (TEMP_ID.NextVal);
```

Si consultamos en este momento el contenido de la tabla:

```
select * from TEMP;
```

veremos que el resultado es:

```
  TEMP_ID
---------
         1
```

- Optimización de las consultas a la base de datos: Existen muchos factores que influyen directamente sobre el rendimiento

de las consultas a la base de datos. Resumirlos aquí sería muy difícil ya que depende en gran medida de la propia base de datos, la arquitectura sobre la que se erige la aplicación, etc. En cualquier caso un pequeño truco que suele funcionar es el que se refiere al orden en el que aparecen las condiciones en la clausula *WHERE* de las sentencias SQL.

Cuando se tiene más de una condición debe ponerse siempre la más restrictiva la última de todas. Por ejemplo, supongamos que tenemos la tabla:

```
VIDEO_TITULO
     VARCHAR2(128)
VIDEO_DISPONIBLE NOT NULL
     NUMBER(1)
```

y lanzamos la siguiente consulta:

```
SELECT count(*)
FROM VIDEO
WHERE VIDEO_TITULO='Armageddon'
     AND VIDEO_DISPONIBLE=1;
```

para calcular el número de ejemplares de una película que están disponibles.

La condición más restrictiva no es *VIDEO_DISPONIBLE=1* ya que pueden existir muchos vídeos que estén disponibles. La condición *VIDEO_TITULO='Armageddon'* en cambio identifica solamente a aquellos ejemplares que haya de esa película. Por lo tanto, para mejorar el rendimiento de la consulta debería ser reescrita de la siguiente forma:

```
SELECT count(*)
FROM VIDEO
WHERE VIDEO_DISPONIBLE=1
     AND VIDEO_ID='Armageddon';
```

- Diferencias entre *VARCHAR2* y *VARCHAR:* A grandes rasgos se puede decir que la diferencia principal que existe entre ambos tipos de datos reside en el hecho de que *VARCHAR2* sólo consumirá el espacio real de la cadena que almacena mientras que *VARCHAR* utiliza todo el espacio que se deduce de la declaración del campo.

```
(1) VIDEO_TITULO VARCHAR2(128)
(2) VIDEO_TITULO VARCHAR(128)
```

Así por ejemplo, si almacenamos en el campo *VIDEO_TITULO* la cadena de texto *'Armageddon'*, en el segundo caso se reservará memoria para guardar los 128 caracteres mientras que en el primero únicamente se utilizará la memoria precisa para almacenar esa cadena de texto.

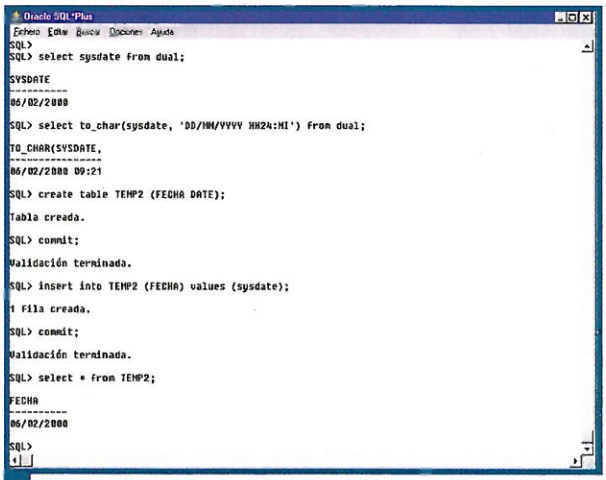


Figura 2.- La palabra reservada *SYSDATE* se utiliza para introducir la fecha y hora actuales.

- Fechas con *Oracle:* En *Oracle* existe una tabla llamada *DUAL* que contiene información diversa como la fecha del sistema. Por ejemplo, lanzando la siguiente consulta:

```
select sysdate
from dual;
```

obtenemos el resultado:

```
SYSDATE
---------
06/02/2000
```

El procedimiento *TO_CHAR* puede utilizarse para dar formato a las fechas tal y como se puede observar en el siguiente ejemplo:

```
select to_char(sysdate,
     'DD/MM/YYYY HH24:MI') from
     dual;
```

El resultado sería:

```
TO_CHAR(SYSDATE,
----------------
06/02/2000 09:21
```

Por último, puedes introducir la fecha y hora actual del sistema utilizando simplemente la palabra reservada *SYSDATE*. Por ejemplo:

```
insert into TEMP2 (FECHA) values
     (sysdate);
```

PREGUNTA

Estoy empezando con *Java* y la verdad es que me cuesta bastante. Quiero hacer un programa que lea un fichero de texto y busque dentro de él una serie de líneas. Cada línea tiene un nombre seguido de dos puntos y después va un número. He estado mirando la documentación de la *API* de *Java* y estoy hecho un lío con eso de los canales de entrada y salida. Os estaría muy agradecido si pudieséis decirme cómo podría hacer algo como esto de la manera más fácil.

También me gustaría que me dijeséis buenas direcciones de *Internet* para aprender *Java*. Muchas gracias.

Roberto Pérez.

RESPUESTA

Estimado Roberto:

Java proporciona un buen número de clases en el paquete *java.io.**, lo que es una gran ventaja para los programadores avanzados pero también provoca confusión a aquellos otros que, como tú, simplemente quieren realizar tareas muy sencillas. A continuación vamos a explicarte paso a paso un pequeño programa que podrá servirte de base para realizar el que tú quieres.

En primer lugar declaramos las variables que van a ser utilizadas en el programa.

```
String linea;
String nombre;
int valor;
int pos_punto;
RandomAccessFile f;
```

La clase *RandomAccessFile*, pertenenciente al paquete *java.io.**, nos permitirá leer el fichero línea a línea. En primer lugar creamos una instancia de la clase *Random AccessFile*:

```
f = new
    RandomAccessFile("C:\\LeerFi-
    chero\\MiFichero.txt","r");
```

A continuación se escribe el bucle principal que se encargará de leer las líneas del archivo de texto.

```
while ((linea = f.readLine()) !=
    null) {
...
}
```

El método *readLine* de la clase *RandomAccessFile* devuelve un objeto del tipo *String* (una cadena de texto) con la línea actual del fichero o *null* si ya se han leído todas las líneas.

El siguiente paso dentro del programa consiste en subdividir la línea en el par *nombre/valor*. Para ello vamos a utilizar dos métodos de la clase *String*: *indexOf* y *substring*.

```
pos_punto = linea.
    indexOf(":");
```

Con la sentencia anterior obtenemos la posición dentro de la cadena de texto donde se sitúa el caracter dos puntos (:). El método *substring* recibe dos parámetros que indican la posición inicial y final que marcan la subcadena que se desea extraer (Los caracteres de las cadenas en *Java* van desde la posición 0 hasta la *longitud-1*). Por consiguiente, las cadenas *nombre* y *valor* pueden ser extraídas de la cada línea haciendo:

```
nombre = linea.substring(0,
    pos_punto);
valor = Integer.parseInt(linea.
    substring(pos_punto+1,
    linea.length()-1),10);
```

Se utiliza el método *length* de una cadena para conocer el número de caracteres que la configuran. La cadena devuelta por *readLine* incluye el propio salto de línea y de ahí que lo evitemos restando una unidad al valor devuelto por *length*. La subcadena obtenida puede convertirse a un número entero utilizando el método *parseInt* de la clase *Integer*.

En cuanto a las direcciones que nos pides te recomendamos que para empezar visites el tutorial de *Sun*:

```
java.sun.com/docs/books/tutorial/
```

Si quieres sitios *Web* en español "*Java* en Castellano" es una de las mejores opciones:

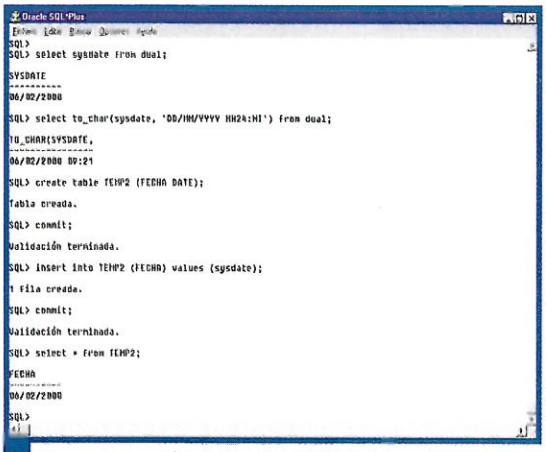


Figura 2.- La palabra reservada *SYSDATE* se utiliza para introducir la fecha y hora actuales.

www.geocities.com/SiliconValley/Lakes/5276/colabora.html

PREGUNTA

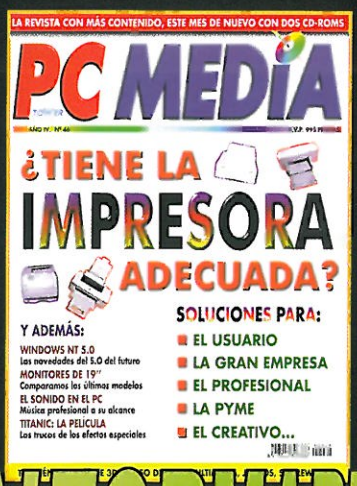
He oído que con *JavaScript* es posible leer ficheros de texto. ¿Puede hacerse esto? ¿Cómo? Quisiera colocar una página *HTML* en mi *Web* que leyera un fichero de texto y mostrara algunos datos.

RESPUESTA

Estimado lector:

Efectivamente sí que se pueden leer ficheros *JavaScript* pero no en el entorno que nos describes en tu pregunta. *Microsoft* ha extendido el uso de *JavaScript* –o deberíamos decir *JScript,* que es como se denomina a la versión que hace *Microsoft*- para que pueda utilizarse como lenguaje de *script* dentro de los sistemas *Windows*. Pero este uso sólo es aplicable a aplicaciones locales, no a las páginas *HTML* de *Internet*. Para hacer eso que tu quieres tendrás que recurrir a un programa *CGI* o a cualquier otra tecnología de servidor, como por ejemplo las páginas *ASP, ColdFusion, Java Servlets, Perl,* etcétera.

¿NECESITAS



LA MEJOR REVISTA DE INFORMÁTICA

LA MEJOR REVISTA PARA USUARIOS DE INTERNET

INFORMARTE,

NAVEGAR,

LA MEJOR REVISTA DE JUEGOS

JUGAR,

LA MEJOR REVISTA DE PROGRAMACIÓN

PROGRAMAR?

TOWER COMMUNICATIONS
C/ Aragoneses, 7 28108 Alcobendas (Madrid)
Tlfno. (91) 661.42.11 Fax. (91) 661.43.86

e-business

Descubra cómo sus aplicaciones pueden jugar en equipo.
Consiga un CD de demostración sin cargo alguno y folleto informativo sobre las soluciones de Comercio Electrónico de IBM en www.software.ibm.com/ec/spainfo o en el 900 100 400 de lunes a viernes de 9 a 19 h.

¿Trabajan todas sus aplicaciones en equipo?

Si su empresa quiere competir en un nuevo campo de juego, -el Comercio Electrónico-, usted puede conseguirlo: conecte entre sí sus aplicaciones existentes y después también a Internet. Con MQSeries de IBM, sus sistemas de pedidos, envíos, inventarios, finanzas y otros procesos de gestión podrán trabajar juntos optimizando sus servidores para el comercio electrónico, sin necesidad de escribir nuevos códigos de conectividad. MQSeries unifica sus aplicaciones, se encarga de las interfaces de programación y soporta Java, COBOL, C++, Lotus Notes y SAP. Además, está disponible en 25 plataformas, incluyendo Windows NT, HP-UX, Sun Solaris e IBM. MQ Series le ayuda a garantizar la entrega de mensajes, incluso cuando fallan los programas o caen las redes. Si usted quiere estar en el campo del comercio electrónico, MQSeries pondrá su negocio en juego.

IBM

Soluciones para nuestro pequeño mundo

MQSeries es marca registrada de IBM Corp. Se reconocen todas las marcas © 1998 IBM.